

(1) Publication number : 2003-131772
(43) Date of publication of application : 09.05.2003

G06F	3/00
G06F	3/16
G10L	15/00
G10L	15/28

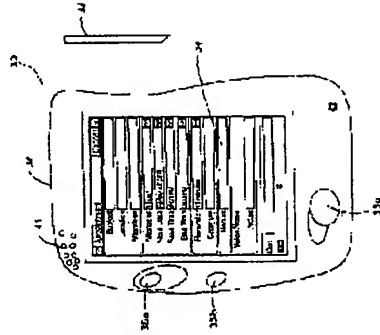
(30)Priority

Priority number : 2001 289041	Priority date : 04.05.2001	Priority country : US
2001 960233	20.09.2001	US
2002 117141	05.04.2002	US

(57)Abstract:

PROBLEM TO BE SOLVED: To improve a method for providing speech recognition and so on through a client/server architecture such as the Internet and so on.

SOLUTION: Documents of a markup language executed on a client device in a client/server system includes instructions indicating a grammar for association with input entered through the client device.



06.05.2005

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

(11)特許出願公開番号

(11)特許出願公開番号
特開2003-131772
(P2003-131772A)

(43)公開日 平成15年5月9日(2003.5.9)

識別記号

601

320

(21)出願番号 特願2002-131950(P2002-131950)

.....

(22)出願日 平成14年5月7日(2002.5.7)

(31)優先權主張番号 60/289,041

(32)優先日 平成13年5月4日(2001.5.4)

(33)優先權主張国 米国 (US)

(31)優先権主張番号 09/960,233

(32) 優先日 平成13年9月20日(2001.9.20)

(33) 優先權主張国 米国 (US)

(31)優先權主張番号 10/117, 141

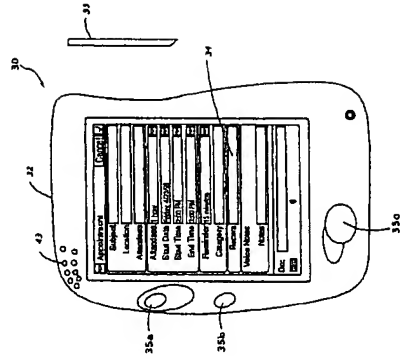
(32)優先日 平成14年4月5日(2002.4.5)

(33) 優先權主張國 米國 (US)

(54) 【発明の名称】 Webで使用可能な認識のためのマークアップ言語拡張部

【課題】 インターネットなどのサーバ/クライアント
アーキテクチャで音声認識等を提供するための方法を改
進する。

【解決手段】 クライアント／サーバシステム中のクライアントデバイスで実行するためのマークアップ言語の文書が、クライアントデバイスを介して入力された入力と関連付けられたための文法を示す命令を含む。



【特許請求の範囲】
 【請求項1】 クライアント／サーバシステム中のクライアントデバイスで実行するためのマークアップ言語を有するコンピュータ読取り可能媒体であって、前記マークアップ言語が、クライアントデバイスを介して入力された入力に関連付けるための文法を示す命令を含むことを特徴とするコンピュータ読取り可能媒体。

【請求項2】 文法を示す前記命令が、前記文法の場所の参照を提供することを特徴とする請求項1に記載のコンピュータ読取り可能媒体。

【請求項3】 前記マークアップ言語が、音声認識、手書き認識、ジェスチャ認識、および視覚認識のうち1つのためのものであることを特徴とする請求項1に記載のコンピュータ読取り可能媒体。

【請求項4】 文法を示す前記命令が、音声認識のための文法の参照を含むことを特徴とする請求項3に記載のコンピュータ読取り可能媒体。

【請求項5】 前記マークアップ言語が、認識された音声に関連する認識結果を受け取り、前記結果を前記クライアントデバイスにあるデータフィールドに関連付けるための命令を含むことを特徴とする請求項1に記載のコンピュータ読取り可能媒体。

【請求項6】 認識結果を受け取るための前記命令が、前記認識結果を複数のデータフィールドに関連付けることを特徴とする請求項8に記載のコンピュータ読取り可能媒体。

【請求項7】 前記マークアップ言語が、前記音声認識された命令を含むように指示するための命令を含むことを特徴とする請求項1に記載のコンピュータ読取り可能媒体。

【請求項8】 前記入力テキスト列を含むことを特徴とする請求項1に記載のコンピュータ読取り可能媒体。

【請求項9】 前記マークアップ言語が、前記テキスト列を他の構成要素から受け取るための命令を含むことを特徴とする請求項8に記載のコンピュータ読取り可能媒体。

【請求項10】 前記マークアップ言語が、前記テキスト列を処理するために前記文法を関連付けることを特徴とする請求項9に記載のコンピュータ読取り可能媒体。

【請求項11】 実施されたときに、入力データ用のフィールドを有するWebサーバからマークアップ言語を受け取るステップと、ユーザから前記フィールドに関する入力を受け取るステップと、

前記入力を示すデータおよび処理用の文法を送信するステップとを含むステップを実行することによってコンピュータに情報処理をさせるコンピュータ読取り可能命令を含むことを特徴とするコンピュータ読取り可能媒体。

【請求項12】 前記指示が、前記文法の場所の参照を提供することを特徴とする請求項11に記載のコンピュータ読取り可能媒体。

【請求項13】 前記指示が、音声認識用の言語の参照を含むことを特徴とする請求項12に記載のコンピュータ読取り可能媒体。

【請求項14】 クライアント／サーバネットワークでの認識のための方法であって、クライアントワーク上のアドレッシングと異なるネットワーク上のアドレッシングと異なるWebサーバから、ネットワークに接続されたクライアントデバイス上に入力データ用のフィールドを有するマークアップ言語ページを受け取るステップと、

前記クライアントデバイス上のフィールドに関するユーザから入力を受け取るステップ、および前記クライアントデバイス上にあるデータフィールドに関連付けるための命令を含むことを特徴とする請求項14に記載のコンピュータ読取り可能媒体。

【請求項15】 前記クライアントデバイス上にあるデータフィールドに関連付けるための命令を含むことを特徴とする請求項15に記載のコンピュータ読取り可能媒体。

【請求項16】 認識された命令を含むことを特徴とする請求項16に記載のコンピュータ読取り可能媒体。

ークアップ言語が、聞き取れるようにプログラミングするための属性または方法を有するオブジェクトモデルを示す命令を含むことを特徴とするコンピュータ読取り可能媒体。

【請求項23】 前記マークアップ言語が、HTML、XHTML、cHTML、XML、およびWMLのうち1つを含むことを特徴とする請求項22に記載のコンピュータ読取り可能媒体。

【請求項24】 前記マークアップ言語が、スクリプトインク言語を含むことを特徴とする請求項23に記載のコンピュータ読取り可能媒体。

【請求項25】 前記マークアップ言語が、合成されたマークアップ言語を含むことを特徴とする請求項22に記載のコンピュータ読取り可能媒体。

【発明の詳細な説明】
 【0001】

【発明の属する技術分野】 本発明は、インターネットなどのワイドエリアネットワークを介した情報へのアクセスに関する。より詳細には、本発明は、様々な方法を使用してクライアント側で情報および制御の入力を可能にする、Webで利用可能な認識に関する。

【0002】

【従来の技術】 パーソナルインフォメーションマネージャ(PIM)などの小型コンピュータインテリジェントデバイス、および携帯電話は、人々の日常生活の中でますます頻繁に使用されている。これらのデバイスを動作させるマイクロプロセッサに現在使用できる処理能力が向上するにつれて、これらのデバイスの機能も向上し、場合によっては統合される。たとえば携帯電話の多くは、インターネットへのアクセスおよびブラウザ機能が可能であり、さらにアドレス、電話番号などの個人情報記録をしておくのに使用することもできる。

【0003】 これらのコンピュータインテリジェントデバイスが、インターネットのブラウザに使用され、または他のサーバ/クライアントアプリケーションで実行されることになり、情報とコンピュータインテリジェントデバイスに入力する必要がある。残念ながら、これらのデバイスは、持ち運びやすいようにできる限り小型であることが望ましく、コンピュータインテリジェントデバイスのブラウザ上では利用可能な表面積が限られているため、すべてのアルファベット文字を独立したボタンとする従来のキーボードは、通常不可能である。

【0004】 近年では、VoiceXML(voice extensible markup language)を使用するよう音声ブラウザが、電話のみを使用してインターネットコンテンツにアクセスできるよう改良されてきた。このアプリケーションでは、ドキュメントサーバ(たとえば、Webサーバ)は、VoiceXMLインテリジェントブラウザを介してクライアントからの要求を処理する。Webサーバはこれに responding Voice

XMLドキュメントを作成することが可能であり、これらのドキュメントは、VoiceXMLインテリジェントブラウザによってユーザが聞き取れるように処理される。ユーザは、音声認識を介した音声コマンドを使用して、Webをナビゲートすることができる。

【0005】

【発明が解決しようとする課題】 VoiceXMLは、フロー制御タグを備えたマークアップ言語であるが、フロー制御は、イベントスクリプトおよび分離スクリプトを含むHTML(Hyper Text Markup Language)フロー制御モデルに従うものではない。むしろ、VoiceXMLは一般に、特に電話ベースの音声専用対話に好適な形態解釈アルゴリズムを含み、一般にユーザから取得された情報は、システムまたはアプリケーションの制御下に置かれる。グラフィカルユーザインターフェースも提供されているクライアントサーバ関係で使用できるアプリケーションに、VoiceXMLを直接組み込むには、開発者が、VoiceXML用のものとHTML(または同様のもの)を使用するものであり、それぞれが異なるフロー制御モデルに従っている。2つの形式のWebブラウザをサポートする必要がある。

【0006】 したがって、インターネットなどのサーバ/クライアントアプリケーションで音声認識を提供するのに使用されるアプリケーションおよび方法を改善することが引き続き求められている。音声認識用のオーサリングツールは、PIMや電話などの小型のコンピュータインテリジェントデバイスに容易に適用できるべきである。前述の欠点のうち1つ、いくつか、あるいはすべてに対処するWebブラウザのアーキテクチャおよび方法が特に求められている。

【0007】

【課題を解決するための手段】 クライアント/サーバシステム中のクライアントデバイスで実行するためのマークアップ言語には、クライアントデバイスを介して入力された入力データに関連付けるための文法を示す命令が含まれる。

【0008】 この拡張部(extension)、およびデバイス、入力データ用のフィールドを有するWebサーバからマークアップ言語ページを受け取るための命令を実行することができる。次いでクライアントデバイスは、そのフィールドに関連するコールから入力データを受け取り、そのデータおよび認識のための文法の指示を、典型的には処理用にリモート位置に配置された認識サーバに送信する。

【0009】 認識サーバは、認識を実行するために、入力データおよび文法の指示を受け取るための命令を実行することができ、次いで、認識の結果をさらに処理するために、クライアントデバイスまたはWebサーバに、クライアントデバイスまたはWebサーバに

【特許請求の範囲】
 【請求項1】 クライアント／サーバシステム中のクライアントデバイスで実行するためのマークアップ言語を有するコンピュータ読取り可能媒体であって、前記マークアップ言語が、クライアントデバイスを介して入力された入力に関連付けるための文法を示す命令を含むことを特徴とするコンピュータ読取り可能媒体。

【請求項2】 文法を示す前記命令が、前記文法の場所の参照を提供することを特徴とする請求項1に記載のコンピュータ読取り可能媒体。

【請求項3】 前記マークアップ言語が、音声認識、手書き認識、ジェスチャ認識、および視覚認識のうち1つのためのものであることを特徴とする請求項1に記載のコンピュータ読取り可能媒体。

【請求項4】 文法を示す前記命令が、音声認識のための文法の参照を含むことを特徴とする請求項3に記載のコンピュータ読取り可能媒体。

【請求項5】 前記マークアップ言語が、認識された音声に関連する認識結果を受け取り、前記結果を前記クライアントデバイスにあるデータフィールドに関連付けるための命令を含むことを特徴とする請求項1に記載のコンピュータ読取り可能媒体。

【請求項6】 認識結果を受け取るための前記命令が、前記認識結果を複数のデータフィールドに関連付けることを特徴とする請求項8に記載のコンピュータ読取り可能媒体。

【請求項7】 前記マークアップ言語が、前記音声認識された命令を含むように指示するための命令を含むことを特徴とする請求項1に記載のコンピュータ読取り可能媒体。

【請求項8】 前記入力テキスト列を含むことを特徴とする請求項1に記載のコンピュータ読取り可能媒体。

【請求項9】 前記マークアップ言語が、前記テキスト列を他の構成要素から受け取るための命令を含むことを特徴とする請求項8に記載のコンピュータ読取り可能媒体。

【請求項10】 前記マークアップ言語が、前記テキスト列を処理するために前記文法を関連付けることを特徴とする請求項9に記載のコンピュータ読取り可能媒体。

【請求項11】 実施されたときに、入力データ用のフィールドを有するWebサーバからマークアップ言語を受け取るステップと、ユーザから前記フィールドに関する入力を受け取るステップと、

前記入力を示すデータおよび処理用の文法を送信するステップとを含むステップを実行することによってコンピュータに情報処理をさせるコンピュータ読取り可能命令を含むことを特徴とするコンピュータ読取り可能媒体。

【請求項12】 前記指示が、前記文法の場所の参照を提供することを特徴とする請求項11に記載のコンピュータ読取り可能媒体。

【請求項13】 前記指示が、音声認識用の言語の参照を含むことを特徴とする請求項12に記載のコンピュータ読取り可能媒体。

【請求項14】 クライアント／サーバシステム中のクライアントデバイスで実行するためのマークアップ言語を有するコンピュータ読取り可能媒体であって、前記マークアップ言語が、認識された命令を含むことを特徴とする請求項14に記載のコンピュータ読取り可能媒体。

【請求項15】 前記クライアントデバイス上にあるデータフィールドに関連付けるための命令を含むことを特徴とする請求項15に記載のコンピュータ読取り可能媒体。

【請求項16】 認識された命令を含むことを特徴とする請求項16に記載のコンピュータ読取り可能媒体。

【請求項17】 前記クライアントデバイス上にあるデータフィールドに関連付けるための命令を含むことを特徴とする請求項17に記載のコンピュータ読取り可能媒体。

CPU50と双方向に結合された不揮発性読取り／書込ミランダムアクセスメモリ記憶装置54に格納される。ランダムアクセスメモリ(RAM)54は、CPU50によって実行される命令用の揮発性記憶域、およびレジスタ値など一時データ用記憶域を提供する。構成要素のうち他の変数用のフォールト値は、読取り専用メモリ(ROM)58に格納される。ROM58は、移動体30の基本機能および他のオペレーティングシステムのカートリッジ機能(たとえば、ソフトウェア構成要素のRAM54へのローディング)を制御する、デバイス用のオペレーティングシステムソフトウェアを格納するために使用することもできる。

【0013】RAM54は、アプリケーションプログラムMを格納するのに使用されるPC上のハードドライブの機能に類似した方法で、コード用の記憶域としての役割も果たす。コードを格納するには不揮発性メモリが使用されるが、代替方法として、コードの実行用には使用されない揮発性メモリ内に格納することもできることに留意される。

【0014】無線番号は、CPU50に結合された無線トランシーバ52を介して移動デバイスによって送受信することができる。コンピュータ(たとえばデスクトップコンピュータ)から、または望むなら有線ネットワークから、データと直接ダウンロードするために、オプションの通信インターフェース60を提供することもできる。したがって、インターフェース60は、たとえば、赤外線リンク、モデム、ネットワークカードなどの様々な形態の通信デバイスを含むことができる。

【0015】移動デバイス30は、マイクroフォン29およびアナログデジタル(A/D)変換器37、ならびに記憶装置54に格納されたアプリケーションの認識プログラム(音声、DTMF、手書き、例を挙げると、デバイス30のユーザからの可聴情報、命令、またはコマンド)に答えて、マイクroフォン29が音声信号を発生し、これをA/D変換器37によってデジタル化される。音声認識プログラムは、音声認識中間結果を取得するために、デジタル化された音声信号に対して、正則化および/または特徴抽出機能を実行することができる。音声データは、無線トランシーバ52または通信インターフェース50を使用して、以下で考察し図5のアーキテクチャに示された、リモート認識サーバ204に伝送される。その後認識結果は、提示し(たとえば視覚的および/または聴覚的)、かつ結果としてWebサーバ202(図5)に伝送するために移動デバイス30に戻され、Webサーバ202および移動デバイス30はクライアント/サーバの関係で動作する。同様の処理が、他の入力形に適用される。たとえば手書き入力、デバイス30上で事前処理によって、または事前処理なしで、デジタル化することができる。音声データと同様に、こ

形態の入力を認識するために認識サーバ204に伝送し、認識結果をデバイス30および/またはWebサーバ202のうち少なくとも一つに戻すことができる。同様にして、DTMFデータ、ジェスチャデータ、および推奨データを同じように処理することができる。入力形態に応じて、デバイス30(および以下で論じる他の形態のクライアント)には、視覚入力用のカメラなどの必要ハードウェアが含まれることになる。

【0016】図3は、携帯電話80の例示的実施形態を示す平面図である。電話80にはディスプレイ82およびキーパッド84が含まれる。一般に図2の構成図は図3の電話に適用されるが、他の機能を実行するのに必要な追加の回路を要求することができる。たとえば、図2の実施形態では、電話として動作することが必要なトランシーバが要求されるが、こうした回路は本発明には関係しない。

【0017】前述の携帯または移動コンシューマデバイスに加えて、本発明が、汎用デスクトップコンピュータなどの他の多数のコンシューマデバイスで使用可能であることも理解されたい。たとえば本発明は、身体的能力の制限されたユーザが、フル英数字キーボードなどの他の従来型入力デバイスを操作するのが困難である場合に、テキストをコンピュータまたは他のコンシューマデバイスに入力できるようにするものである。

【0018】本発明は、他の多数の汎用または特定用途向けコンシューマシステム、携帯、または構成で動作可能である。本発明で使用するのに好適な、よく知られたコンシューティングシステム、環境、および/または構成の例には、通常の電話(画面なし)、パーソナルコンピュータ、サーバコンピュータ、ハンドヘルドまたはラップトップデバイス、マルチプロセスシステム、マイクロプロセッサベースシステム、セットトップボックス、プログラム可能大消費電子製品、ネットワークPC、ミニコンピュータ、メインフレームコンピュータ、上記システムまたはデバイスのいずれかを含む分散コンシューティング環境などが含まれるが、これらに限定されるものではない。

【0019】次に、図4に示した汎用コンピュータ120について、簡単に説明する。ただし、コンピュータ120も好適なコンシューティング環境の一例に過ぎず、本発明の使用法または機能の範囲に關して何らかの制限を示唆するものではない。さらにコンピュータ120は、本明細書に例示された構成要素のいずれか1つまたははいずれかの組合せに關して、なんらかの依存性または要求条件を有するものとして解釈されるものでもない。【0020】本発明は、コンピュータによって実行されるプログラムモジュールなどの、コンピュータ実行可能命令の一般的な文脈で説明することができる。一般に、プログラムモジュールには、特定のタスクを実行するか、

または特定の抽象データ型を実施する、ルーチン、プログラム、オブジェクト、構成要素、データ構造などが含まれる。本発明は、通信ネットワークを介してリンクされたリモート処理デバイスによってタスクが実施される。分散コンシューティング環境で実施することは可能である。分散コンシューティング環境では、プログラムモジュールを、メモリ記憶デバイスを含む、ローカルおよびリモートの両方のコンピュータ記憶媒体に配置することが可能である。プログラムおよびモジュールによって実行されるタスクについては、以下で図を用いて説明する。当分野の技術者であれば、記述および図面を、任意の形態のコンピュータ読取り可能媒体上に書き込み可能な処理装置実行可能命令として実施することができ

る。【0021】図4を参照すると、コンピュータ120の構成要素には、処理ユニット140、システムメモリ50、およびシステムメモリを含む様々なシステム構成要素を処理ユニット140に結合するシステムバス141が含まれるが、これらに限定されるものではない。システムバス141は、メモリバスまたはメモリ制御装置、周辺バス、および様々なバスアーキテクチャのいずれかを使用するローカルバスを含む、いくつかのタイプのバスストラクチャのうちのいずれかであってよい。例を挙げると、こうしたアーキテクチャには、Industry Standard Architecture (ISA)バス、Universal Serial Bus (USB)、Micro Channel Architecture (MCA)バス、拡張ISA (EISA)バス、Video Electronics Standards Association (VESA)ローカルバス、およびメザンバスと呼ばれるPeripheral Component Interconnect (PCI)バスが含まれるが、これらに限定されるものではない。コンピュータ120は、典型的には様々なコンピュータ読取り可能媒体を含む。コンピュータ読取り可能媒体は、コンピュータ120がアクセスすることのできる任意の使用可能媒体であってよく、揮発性および不揮発性媒体、取外し可能および取外し不能媒体の両方を含む。例を挙げると、コンピュータ読取り可能媒体は、コンピュータ記憶媒体および通信媒体を含むことができるが、これらに限定されるものでは

送信することができる。【0010】

【発明の実施の形態】同じ内容を実施するためのWebページの認識およびメソッドのアーキテクチャについて説明する前に、そのアーキテクチャで機能できる一般的なコンピュータシステムについて説明する方が有用であろう。図1を参照すると、データ管理デバイス(PIM、PDAなど)の形態例が30に示されている。ただし、本発明は、以下で論じる他のコンピュータシステム、具体的に言えば、入力ポタブル用などの装置に限られているコンピュータデバイスを使用しても実施可能であることが念及される。たとえば、電話および/またはデータ管理デバイスも、本発明から恩恵を受けることになる。こうしたデバイスは、既存の携帯用個人情報管理デバイスおよび他の携帯用電子デバイスに比べて、機能強化されたユーティリティを有するものとなり、こうしたデバイスの機能およびコンパクトなサイズにより、ユーザに対して常に持ち運ぶように奨励しやすくなる。したがって、本明細書に記載されたアーキテクチャの範囲は、本明細書に示した例示的なデータ管理またはPIMデバイス、電話、またはコンピュータの開示によって限定されることを意図するものではない。

【0011】データ管理移動デバイス30の例示的形態が、図1に示されている。移動デバイス30は、ハウジング32を含む、スタイル33と一緒に使われる後触感応性のディスプレイ面を使用するディスプレイ34を含むユーザインターフェースを有する。スタイル33は、フィールドを選択するために指定された位置位置では、フィールド34を跨すかまたは接触するため、カーソルの開始位置を選択的に移動するため、あるいはその他の方法でジェスチャまたは手書きなどのコマンド情報を提供するために、使用される。別法では、または加えて、デバイス30上にナビゲーション用の1つまたは複数のボタン35を含めることができる。さらに、回転ホイール、ローラなどの他の入力メカニズムを提供することもできる。ただし、本発明は、これらの入力メカニズム形態によって限定されるものではないことに留意されたい。たとえば他の入力形態には、コンピュータジョーンを介するなどの投機的入力を含むことができる。

【0012】次に図2を参照すると、移動デバイス30を含む通信要素の構成図が示されている。中央処理ユニット(CPU)50は、ソフトウェア制御機能を実施する。CPU50がディスプレイ34に結合されているため、制御ソフトウェアによって生成されたテキストおよびグラフィックディスプレイ34に提示される。スピーカ43は、可聴出力を提供するために、典型的にはデジタルアナログ変換器59を使用してCPU50に結合することができる。ユーザによってダウンロードされ、移動デバイス30に入力されたデータは、こ

ーフエースを介してシステムバス141に接続され、磁気ディスクドライブ171および光ディスクドライブ175は、典型的には、インターフェース170などの取外し可能メモリーインターフェースによってシステムバス141に接続される。

[0025] 上記で考案し図4に示したドライブおよびその関連付けられたコンピュータ記憶媒体は、コンピュータ120に、コンピュータ制御可能な命令、データ構造、プログラムモジュール、および他のデータの記憶媒体を提供する。たとえば図4では、ハードディスクドライブ161がオペレーティングシステム164、アプリケーションプログラム165、他のプログラムモジュール166、およびプログラムデータ167を格納しているように図示されている。これらの構成要素は、オペレーティングシステム154、アプリケーションプログラム155、他のプログラムモジュール156、およびプログラムデータ157と同じである場合も異なる場合も、いずれも可能であることに留意されたい。本明細書では、オペレーティングシステム164、アプリケーションプログラム165、他のプログラムモジュール166、およびプログラムデータ167には、少なくとも異なるコピーであることを示すために異なる番号が与えられている。

[0026] ユーザは、キーボード182、マイクロプロセッサ183、およびマウス、トラックボール、またはタッチパッドなどのポインティングデバイス181などの入力デバイスを通じて、コマンドおよび情報をコンピュータ120に入力することができる。他の入力デバイス(図示せず)には、ジョイスティック、ゲームパッド、衛星放送用アンテナ、スキャナ、などが含まれる。これらおよび他の入力デバイスは、システムバスに結合され、ユーザ入力インターフェース180を介して処理ユニット140に接続される場合が多いが、パラレルポート、ゲームポート、またはUniversal Serial Bus (USB) などの他のインターフェースおよびバス構造によって接続することもできる。モニタ184または他のタイプのディスプレイデバイスも、ビデオインターフェース185などのインターフェースを通じてシステムバス141に接続される。コンピュータ171、ならびにCD-ROMまたは他の光ディスク媒体などの取外し可能な非揮発性光ディスク176との間で読取りまたは書き込みを行う光ディスクドライブ175が示されているが、これは例として示したものにすぎない。例示的な動作環境で使用することのできる他の取外し可能/取外し不能、揮発性/非揮発性コンピュータ記憶媒体には、磁気テープカセット、フラッシュメモリーカード、デジタル汎用ディスク、デジタルビデオテープ、リッドステートRAM、ソリッドステートROM、などが含まれるが、これらに限定されるものではない。ハードディスクドライブ161は、典型的には、インターフェース160などの取外し不能メモリーインタ

拡張部を使用することで、Webサーバ202でのオーサリングが容易になり、現在存在しているレガシーアプリケーションを音声認識を含むように修正することも容易となる。

[0031] 一般に、デバイス30は、Webサーバ202が提供するHTMLページ、スクリプトなどを実行する。例を挙げると、音声認識が要求される場合、可聴信号をデジタル化することができる音声データ、または前述のようにデバイス30によって可聴信号が即時処理される音声特徴が、音声認識時に使用するための文法または音韻モデルの指示と共に、認識サーバ204に提供される。認識サーバ204の実施は多くの形態を取ることが可能であり、そのうちの1つが図示されているが、一般には認識装置211が含まれる。認識の結果は、所望されるかまたは適切であれば、ローカルで提示するためにデバイス30に渡される。認識および使用される情報をWebサーバ202に送信してさらに処理し、必要であればさらにHTMLページ/スクリプトを受け取る。

[0032] 図5に示されるように、デバイス30、Webサーバ202、および認識サーバ204は、一般にネットワーク205を介して接続されるが、本明細書ではインターネットなどのワイドエリアネットワークを介して接続されており、別々にアドレス指定可能である。したがって、これらのデバイスはいずれも物理的に相互に隣接させる必要はない。具体的に言えば、Webサーバ202が認識サーバ204を含む必要はない。この方式では、Webサーバ202でのオーサリングは、作者が認識サーバ204の複雑な内容を知る必要なく、意図されているアプリケーションに焦点をあてることが可能である。むしろ認識サーバ204は、単独で設計しネットワーク205に接続することが可能であり、それによって、Webサーバ202で要求される更新なしに、更新および改良することが可能である。以下で論じよう。Webサーバ202は、動的にクライアント側のマークアップおよびスクリプトを生成することができるオーサリング機構を含むこともできる。他の実施形態では、Webサーバ202、認識サーバ204、およびクライアント30を、実施マシンの機能に応じて組み合わせてることができる。たとえばパーソナルコンピュータを含むコンピュータ、所望であれば、Webサーバ202および認識サーバ204を単一のマシンに組み込むことができる。

[0033] 本発明の一態様は、クライアントデバイスユーザから入力データを取得するように構成された拡張部を有するマークアップ言語ページをサーバから受け取る。

[0033] 本発明の一態様は、クライアントデバイスユーザから入力データを取得するように構成された拡張部を有するマークアップ言語ページをサーバから受け取る。

[0033] 本発明の一態様は、クライアントデバイスユーザから入力データを取得するように構成された拡張部を有するマークアップ言語ページをサーバから受け取る。

ーフエースを介してシステムバス141に接続され、磁気ディスクドライブ171および光ディスクドライブ175は、典型的には、インターフェース170などの取外し可能メモリーインターフェースによってシステムバス141に接続される。

[0025] 上記で考案し図4に示したドライブおよびその関連付けられたコンピュータ記憶媒体は、コンピュータ120に、コンピュータ制御可能な命令、データ構造、プログラムモジュール、および他のデータの記憶媒体を提供する。たとえば図4では、ハードディスクドライブ161がオペレーティングシステム164、アプリケーションプログラム165、他のプログラムモジュール166、およびプログラムデータ167を格納しているように図示されている。これらの構成要素は、オペレーティングシステム154、アプリケーションプログラム155、他のプログラムモジュール156、およびプログラムデータ157と同じである場合も異なる場合も、いずれも可能であることに留意されたい。本明細書では、オペレーティングシステム164、アプリケーションプログラム165、他のプログラムモジュール166、およびプログラムデータ167には、少なくとも異なるコピーであることを示すために異なる番号が与えられている。

[0026] ユーザは、キーボード182、マイクロプロセッサ183、およびマウス、トラックボール、またはタッチパッドなどのポインティングデバイス181などの入力デバイスを通じて、コマンドおよび情報をコンピュータ120に入力することができる。他の入力デバイス(図示せず)には、ジョイスティック、ゲームパッド、衛星放送用アンテナ、スキャナ、などが含まれる。これらおよび他の入力デバイスは、システムバスに結合され、ユーザ入力インターフェース180を介して処理ユニット140に接続される場合が多いが、パラレルポート、ゲームポート、またはUniversal Serial Bus (USB) などの他のインターフェースおよびバス構造によって接続することもできる。モニタ184または他のタイプのディスプレイデバイスも、ビデオインターフェース185などのインターフェースを通じてシステムバス141に接続される。コンピュータ171、ならびにCD-ROMまたは他の光ディスク媒体などの取外し可能な非揮発性光ディスク176との間で読取りまたは書き込みを行う光ディスクドライブ175が示されているが、これは例として示したものにすぎない。例示的な動作環境で使用することのできる他の取外し可能/取外し不能、揮発性/非揮発性コンピュータ記憶媒体には、磁気テープカセット、フラッシュメモリーカード、デジタル汎用ディスク、デジタルビデオテープ、リッドステートRAM、ソリッドステートROM、などが含まれるが、これらに限定されるものではない。ハードディスクドライブ161は、典型的には、インターフェース160などの取外し不能メモリーインタ

[0024] コンピュータ120は、他の取外し可能/取外し不能、揮発性/非揮発性のコンピュータ記憶媒体を含むこともできる。図4では、取外し不能の非揮発性磁気媒体との間で読取りまたは書き込みを行うハードディスクドライブ161、取外し可能な非揮発性磁気ディスク172との間で読取りまたは書き込みを行う磁気ディスクドライブ171、ならびにCD-ROMまたは他の光ディスク媒体などの取外し可能な非揮発性光ディスク176との間で読取りまたは書き込みを行う光ディスクドライブ175が示されているが、これは例として示したものにすぎない。例示的な動作環境で使用することのできる他の取外し可能/取外し不能、揮発性/非揮発性コンピュータ記憶媒体には、磁気テープカセット、フラッシュメモリーカード、デジタル汎用ディスク、デジタルビデオテープ、リッドステートRAM、ソリッドステートROM、などが含まれるが、これらに限定されるものではない。ハードディスクドライブ161は、典型的には、インターフェース160などの取外し不能メモリーインタ

[0027] コンピュータ120は、リモートコンピュータ194などの1つまたは複数のリモートコンピュータへの論理接続を使用するネットワーク環境で動作可能である。リモートコンピュータ194は、パーソナルコンピュータ、ハンドヘルドデバイス、サーバ、ルータ、ネットワークPC、ピアデバイス、または他の共通ネットワークノードであってよく、典型的にははコンピュータ

取り、クライアントデバイス上でマークアップ言語ページを実行し、入力データ（ユーザから取得した音声、DTMF、手書き、ジェスチャー、またはイメージを示す）および関連付けられた文法をクライアントから遠隔にある終端サーバに伝送し、なにより、クライアントの終端サーバから認識結果を受取るステップを含む、クライアント/サーバシステムで入力データを処理するための方法である。他の態様は、クライアント/サーバシステムのクライアントデバイスで実行するためのマークアップ言語を有するコンピュータ積取り可能媒体であって、このマークアップ言語は、クライアントデバイスを介してマークアップ言語に関連付けられるための文法を示す命令を有する。

【0034】電話80を介したWebサーバ202における、電話80から外部端末または無線電話ネットワーク208への接続がなされ、これが電話80を第三者ネットワーク210に接続する。ゲートウェイ210は、電話80を電話音声ブラウザ212に接続する。電話音声ブラウザ212は、電話インターネットファーストパーティー音声ブラウザ216を提供する。ディアルプラン214は電話音声ブラウザ212に、HTMLページ/スクリプトなどをWebサーバ212はHTMLページ/スクリプトなどに、HTMLページ202から受け取る。さらに重要なことに、HTMLページ/スクリプトは、デバイス30に提供されるHTTMLページ/スクリプトと同様の形式である。この方式では、Webサーバ202がデバイス30および電話のGUMを別々にサポートする必要があるが、または標準のGUMクライアントを別々にサポートする必要もない。むしろ、通常のマークアップ言語を適用することができ、送られる可搬信号からの音声認識が、ネットワーク202によって5または専用回線207のいずれかを介して、たとえばTCP/IPを使用して、音声ブラウザ216から認識サーバ204に提供される。Webサーバ202、認識サーバ204、および電話音声ブラウザ212は、図4に示された汎用デスクトップコンピュータなどの任意の好適なコンピューティング環境で実施することができ

【0035】ただし、DTMF認識が使用される場合、この形態の認識は一般に、認識サーバ204ではなくメディアサーバ214で実行されることに留意されたい。言い換えれば、DTMF文法はメディアサーバによって使用される。

【0036】前述のように、本発明の一態様は、クライアント/サーバアーキテクチャに認識を実現する制御処理および/またはオブジェクトを含めるための、HTML、XHTML、cHTML、XML、WMLなどのマークアップ言語の拡張部を含む、または任意の他のSGMLやXML型言語の拡張部を備えている。この方式では、作者が、こうしたアーキテクチャで使用される有力なWebページ、こうしたアーキテクチャを備えている、この方式では、作者が、こうしたアーキテクチャで使用される有力なWeb

起動すると、関数「verify」が呼び出される。またはスクリプト部分262で実行される。関数「verify」は、各クレジットカード（Visa、MasterCard、およびAmerican Express）のカード番号が適切な長さであるかどうかを確認する。

【0040】図8は、音声認識を使用してWebサーバ204に提供されるクレジットカード情報を取得するために、図6のGUIと同じものを生成するクライアント208〜図16に關して以下で論じ、説明する技術は、手書き認識、音声認識、およびイメージ認識で同様に着用可能であることを理解されたい。

[0041] 一般に、拡張部（一般に「タグ」とも呼ばれる）は、関連付けられた任意およびDOMオブジェクトのプロパティを備えたXML要素と、認識ドキュメントフェーズを適用するためのソースアップロードキーマンifestファイルに関連して使用することであり、また、そのソースコードの一部分として使用することもできる。拡張部は、HMLまたはDHTMLなどのマークアップ言語との互換性を確保するために、特定の用途のために設計されたものであり、その中には、XMLスキーマやスタイルシートなどがある。拡張部の目的は、標準的なXMLスキーマで定義されていないデータ構造をサポートすることである。拡張部は、XMLスキーマを使用して、等しく有効に使用することができる。拡張部は、構文規則に従って新しい機能オブジェクトモデルまたは他のオブジェクトモデルを提供する。各要素については付録Bに記載されているように、一般的には、プロパティ、メンバシップ、およびその他の「子」要素を含むことができる。

【0042】ここで、拡張部は、ブラウザが実行されるデバイス上の機能に従って、2つの異なる「mode」で解釈されることに留意されたい。第1のmode「object mode」では、全機能は使用可能である。アプリケーションによる拡張部のプログラム操作は、たとえば、XHTMLブラウザのJavaScriptインタプリタ、またはWMLブラウザのWMLScriptインタプリタなど、デバイス上のブラウザによって使用可能にされるどのような機構によっても実行される。そ

のため、証券部のコアプログラマーおよびメソッドのマスターまたはクライアント側を要する以上、これは、デバインスツールセットに追加されるようなプログラムシステムまたはクライアントに存在する。object mode機構によっても動作される。object modeは、音声対話を介して、より複雑なクライアント制御、即ちダイアログ作者に与えるためのより優れた機能を提供することができる。本明細書で使用される場合、クライアントおよびスクリプトをサポートしているブラウザは「アップレベルブラウザ (uplevel browser)」と呼ばれる。この形態のブラウザは、拡張

owser)」と呼ばれる。この形態のブラウザは、並

(10)	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463	464	465	466	467	468	469	470	471	472	473	474	475	476	477	478	479	480	481	482	483	484	485	486	487	488	489	490	491	492	493	494	495	496	497	498	499	500	501	502	503	504	505	506	507	508	509	510	511	512	513	514	515	516	517	518	519	520	521	522	523	524	525	526	527	528	529	530	531	532	533	534	535	536	537	538	539	540	541	542	543	544	545	546	547	548	549	550	551	552	553	554	555	556	557	558	559	560	561	562	563	564	565	566	567	568	569	570	571	572	573	574	575	576	577	578	579	580	581	582	583	584	585	586	587	588	589	590	591	592	593	594	595	596	597	598	599	600	601	602	603	604	605	606	607	608	609	610	611	612	613	614	615	616	617	618	619	620	621	622	623	624	625	626	627	628	629	630	631	632	633	634	635	636	637	638	639	640	641	642	643	644	645	646	647	648	649	650	651	652	653	654	655	656	657	658	659	660	661	662	663	664	665	666	667	668	669	670	671	672	673	674	675	676	677	678	679	680	681	682	683	684	685	686	687	688	689	690	691	692	693	694	695	696	697	698	699	700	701	702	703	704	705	706</
------	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-------

21
 ことができる。前述のように、音声データは、ユーザが入力した音声に関連付けられたストリーミングデータを、スク립ト関数「checkFilled」によって全体的に制御される。しかし、文法は、図8に関して上記で述べたのと同じ文脈を使用して起動され、音声データおよび使用した文法の指示が認識サーバ204に提供される。同様に、認識サーバ204から受け取られる出力は、クライアントのフィールドに関連付けられる（本明細書では電話音声ブラウザ212）。

10
 【0064】 一般に音声専用アプリケーションに固有の他の特徴は、音声認識された場合のユーザに対する指示である。図8などのマルチメディアアプリケーションでは、fonNoRecolは、認識していないことを示す際に表示されたフィールドにnull値を入れるだけであり、ユーザは情報を入力されたかまたは話されたときに何らかの制御を行うことができる。言い換えれば、システムはユーザの話を開始させるか、そうでなければユーザに話を開始するように指示することが可能であるが、ユーザは最初に要求された以上の情報を提供することができる。これが「主導権混合 (mixed initiative)」の一例である。一般にこの形態の対話 (dialog interaction) では、ユーザが対話の主権権をシステムと共有することができる。ユーザがプロンプトで要求された以上の情報を提供する場合、前述および以下で詳細に論じられる例に加え、ユーザは指示されていない場合にもタスクを切り換えることができる。

30
 【0069】 図11および図12の例では、do_fieldとして示された文法に、文法「g_card_types」、「g_card_num」、および「g_expiry_date」に関連付けられた情報が含まれる。この例では、fonRecolで示されるように認識された音を受け取る、電話音声ブラウザ212が電話80から受け取った音声データおよび文法「do_field」を使用して、認識サーバ204に送信し、音声データからの認識された任意のすべてのフィールドに関する値を関連付けることができる。言い換えれば、認識サーバ204から取得し含む関数「handle」が、呼び出されるかまたは実行される。言い換えれば、認識サーバ204から取得した結果には、各フィールドに関する指示も含まれるというものである。この情報が解析され、405に指定されたbind規則に従って、対応するフィールドに関連付けられる。図5に示されるように、認識サーバ204は構文解析装置309を含むことができる。

40
 【0070】 図7、図8、図9、図10、図11、および図12では、非常に似通ったWeb開発フレームワークが使用されている。それぞれの場合のデータ指示も非常に似通っている。さらに、データ指示とフロー制御を分離することで、異なるアプリケーション間 (システム主導および主導権混合) または異なるモダリティ (modality) 間 (GUI、Webベース、音声専用、およびマルチメディア) で、最大の再利用性を可能にしている。さらにこれによって、電話がデバイス30と同一線のディスプレイおよび機能を含む場合に、音声専用動

作から電話を介してマルチメディア動作へという、自然な拡張も可能にしている。付録Aでは、上記で述べた制御およびオブジェクトについて、さらに詳細に説明する。

【0071】 上記のように、アップレベルブラウザは、認識結果を割り当てするために、上記の例で関数「handle」を呼び出すことを含む様々な必要事項を実行するために、スク립ティングを使用することができる。前述の実施形態および付録Aの2.1.2でさらに述べるように、「bind」要素は認識結果を解析して値を割り当てるものであって、「bind」要素は「reo」要素のサブ要素または子要素である。

10
 【0072】 スクリプティングは有用であるが、たとえばセキュリティの問題により、これが最善のブラウザ実装形態ではないと考えている人が多い。したがって、本発明の他の実施形態または他では、「bind」要素は高位要素 (freco) と同様、であった。他のより豊富なプロパティを備えており、本質的にはスク립ティングせずに、実質上はスク립ティングを模倣することができる。

20
 【0073】 スクリプティングを使用しないか、または以下で論じる本発明の形態を使用しない、高度な対話効果などの以下で論じる機能の中には、ページをWebサーバ202に送ること、新しいページを生成するためにアプリケーション論理を実行すること、およびページをクライアントデバイスに返送することによってのみ達成できるものもある。本発明のこの態様により、プログラマは、サーバのラウンドトリップなしにページのオブジェクトに関するメソッドを呼び出すことができる。

30
 【0074】 前述の実施形態では、「bind」要素は、認識結果を書式またはWebページ内のフィールドに割り当てるための属性「targetElement」および「targetAttribute」のみを有する。他の実施形態では、「bind」要素は「targetMethod」も含んでおり、これがオブジェクトメソッド呼び出し用を追加される。「targetMethod」の使用および機能は、スク립ティングを模倣する上で主要な技法である。たとえば以下の構文を使用して、オブジェクト「OBJ1」の「X」メソッドを呼び出すことができる。

40
 <bind TargetElement = "OBJ1" TargetMethod = "X" ...>

【0075】 本明細書で示される例はHTML/XHTML MLイベント構文に従うものであるが、当分野の技術者であれば、W3C Document Object Model Level 2またはLevel 3イベント定義標準、ECMA Common language Infrastructure (CLI) イベントモデル、Java (登録商標) プログラミング言語イベントモデル、W3C Synchronous Mu

21
 ことができる。前述のように、音声データは、ユーザが入力した音声に関連付けられたストリーミングデータを、スク립ト関数「checkFilled」によって全体的に制御される。しかし、文法は、図8に関して上記で述べたのと同じ文脈を使用して起動され、音声データおよび使用した文法の指示が認識サーバ204に提供される。同様に、認識サーバ204から受け取られる出力は、クライアントのフィールドに関連付けられる（本明細書では電話音声ブラウザ212）。

10
 【0064】 一般に音声専用アプリケーションに固有の他の特徴は、音声認識された場合のユーザに対する指示である。図8などのマルチメディアアプリケーションでは、fonNoRecolは、認識していないことを示す際に表示されたフィールドにnull値を入れるだけであり、ユーザは情報を入力されたかまたは話されたときに何らかの制御を行うことができる。言い換えれば、システムはユーザの話を開始させるか、そうでなければユーザに話を開始するように指示することが可能であるが、ユーザは最初に要求された以上の情報を提供することができる。これが「主導権混合 (mixed initiative)」の一例である。一般にこの形態の対話 (dialog interaction) では、ユーザが対話の主権権をシステムと共有することができる。ユーザがプロンプトで要求された以上の情報を提供する場合、前述および以下で詳細に論じられる例に加え、ユーザは指示されていない場合にもタスクを切り換えることができる。

20
 【0069】 図11および図12の例では、do_fieldとして示された文法に、文法「g_card_types」、「g_card_num」、および「g_expiry_date」に関連付けられた情報が含まれる。この例では、fonRecolで示されるように認識された音を受け取る、電話音声ブラウザ212が電話80から受け取った音声データおよび文法「do_field」を使用して、認識サーバ204に送信し、音声データからの認識された任意のすべてのフィールドに関する値を関連付けることができる。言い換えれば、認識サーバ204から取得し含む関数「handle」が、呼び出されるかまたは実行される。言い換えれば、認識サーバ204から取得した結果には、各フィールドに関する指示も含まれるというものである。この情報が解析され、405に指定されたbind規則に従って、対応するフィールドに関連付けられる。図5に示されるように、認識サーバ204は構文解析装置309を含むことができる。

30
 【0070】 図7、図8、図9、図10、図11、および図12では、非常に似通ったWeb開発フレームワークが使用されている。それぞれの場合のデータ指示も非常に似通っている。さらに、データ指示とフロー制御を分離することで、異なるアプリケーション間 (システム主導および主導権混合) または異なるモダリティ (modality) 間 (GUI、Webベース、音声専用、およびマルチメディア) で、最大の再利用性を可能にしている。さらにこれによって、電話がデバイス30と同一線のディスプレイおよび機能を含む場合に、音声専用動

l t i m e d i a I n t e g r a t i o n L a n g u a g e (S M I L)、および近目増大されるW3C X M Lイベント標準提案を含むが、これらに限定されるものではない、他のイベントインテグレーション、< b i n d > の用法を一般化するのを簡単であることを理解されることが留意されたい。

【0076】図17および図18は、クライアント、具体的にはdown level browser上で実行可能なマークアップ言語のページである。この例では、ユーザが可聴プロンプトを介して飲み物についてたずねられる。次にシステムは、何の飲み物が注文されたかを識別する。「bind」要素はこの認識結果に応じて、宣言処理を使用して実行を指導する。飲み物が確認されると、この書式はすべてのスクリーンがなしでWebサーバ202に返される。

【0077】一般に、図17および図18のマークアップ例には、データセクション350、音声セクション352、およびユーザインタラクションセクション354、356、および358が含まれる。セクション354は、ユーザの希望する飲み物に関する一般問合せから認識結果を受け取り、クリアームまたは砂糖が必要であるかどうかに関する再プロンプト、問合せ、あるいは注文された飲み物の確認のいずれかに向けて、対話型認識フローを送る。セクション355は、セクション356は、クリアームまたは砂糖も注文されたときに認識結果を受け取る。セクション358は、飲み物の確認に関する認識結果を受け取る。セクション360は新しいメッセージングオブジェクト「SMEX」を使用する制御セクションであり、これについては以下で詳細に論じる。

【0078】前述のように、本発明のこの態様の「bind」要素にはオブジェクトメソッド呼び出しが含まれ、これが361でfwelecomeオブジェクトのfstartメソッドが実行されたときに、fwelecomeオブジェクトを再生することによって、図17および図18の例でユーザ対話を開始させる。

【0079】次にユーザは、362で「asked」オブジェクトのfstartメソッドを実行することによって、「Do you want coke, coffee or orange juice?」となす。次に、363で「reco_drink」オブジェクトのfstartメソッドを呼び出すことによって、認識が実行される。

【0080】次にセクション354のマークアップが実行され、認識サーバ204が使用する方法が、XPathステートメント「/drink types」による提供される。この例では、W3CのXPath言語を使用しているが、当分野の技術者であれば、W3C XML照会言語(XQL)を含むことが可能であることが論議でとれない他の標準言語にまで概念を広げることが論議であることを理解されたい。「bind」

要素364で指定されるように、認識サーバ204から受け取った認識結果の値が10より小さい場合、366でプロンプトオブジェクト「frequest」が、続いて368でプロンプトオブジェクト「ask」が実行され、370で認識オブジェクト「reco_drink」が再開される。戻された認識結果が、値が10より大きい「coffee」の場合、372でフィールド「drink」に認識結果の値が割り当てられ、次にユーザは、374で、プロンプトオブジェクト「cream_sugar」によってクリアームまたは砂糖を希望するかどうかに関するプロンプトを受け取る。次に376で、セクション356の認識オブジェクト「reco_cream_sugar」が呼び出される。これは正しい、認識結果の値が10より大きい「coffee」でない場合、378で再度drinkフィールドが割り当てられる。380でプロンプトオブジェクト「confirm」を実行し、その後セクション358の382で認識オブジェクト「fyes」を呼び出すことによって、認識結果の確認が与えられる。ユーザが10より大きい値で「yes」と答えると、384でプロンプトオブジェクト「hshakes」が再生され、その後386で書式がサブミットされる。これは正しい、ユーザが「no」と答えるか、または認識結果の値が10より小さい場合、390でプロンプトオブジェクト「fretry」が実行され、もう一度392でプロンプトオブジェクト「ask」が実行されて、394で認識オブジェクト「reco_drink」が呼び出される。

【0081】前述の例からわかるように、「bind」要素は、セクション354、356、または358に示すようにメソッドの複数呼び出しを可能にする。所望であれば、認識された結果の複数割当てを宣言することもできる。例示された実施形態では、複数の割当ておよびメソッド呼び出しが宣言されると、ドキュメント順で実行される。

【0082】他の実施形態では、メソッド引数を渡すための規定も提供される。言い換えれば、メソッドの中には引数リストが必要なものもある。これは、「arg」サブ要素を使用して達成される。たとえば、以下のマークアップ

```
<bind TargetElement = "obj" TargetMethod =
  "f">argX</arg>X</arg></bind>
```

は、「OB」f(X,Y)と等価であるか、あるいは「OB」fが、パラメータまたは引数「X」および「Y」を使用するメソッド「f」を有するオブジェクトである。

【0083】「bind」要素は、bind要素がどのイベント用であるかを宣言する「event」属性を含むことも可能である。たとえば、以下のマークアップ

```
<bind event = "onNoReco" = TargetElement =
```

"prompt" TargetMethod = "start"/>

は、「onNoReco」イベントが送られたときを意味するものであり、オブジェクト「prompt」の「start」メソッドが呼び出されることになる。図8に関連して上記の例で説明したように、「bind」要素は「reco」要素の子要素として使用する場合と整合性を保たせるために、「bind」要素のデフォルト属性は「onReco」である。

【0084】高位要素としての「bind」要素は、付録2、4項に指定されたいずれのイベントも含むことができる。さらに、「bind」要素は、アクセス可能であってプログラムフローの向きを決めるのに使用できる「status」属性を有する。「onError」イベントが「status」属性を含む。「bind」要素の他のイベントが「status」属性を有する場合、これらにも同様にアクセスできる。

【0085】認識結果の条件をチェックすることに加え、実行予定の現在のドキュメントまたはページでチェックすることができる。具体的に言えば、「test」属性および「value」属性の両方を、格納ドキュメントのルートノードを示す「host」プリミティブを含むように拡張することができる。たとえば、図17、図18を再度参照すると、そこに含まれる例は、セクション354に、ユーザがcoffeeを要求したときにクリアームまたは砂糖を希望かどうかをたずねるための追加処理がある。クリアームまたは砂糖を追加するた

め、したがって、マークアップ「host() /get_drink / drink = coffee」によって指定されるように、セクション356を呼び出すためのフラグは、drinkフィールドが「coffee」の場合に限ってオンになる。

【0086】また、「bind」要素は音声サーバ204からの認識結果およびドキュメント内の値の受取りまたは割当てに適用可能だけでなく、たとえば、クライアントデバイス上で実行中のアプリケーションからのメッセージオブジェクト(本明細書では「smex」で示されている)にも適用可能であることに留意されたい。図17および図18の例では、クライアントデバイス上で実行中の電話アプリケーションが呼を待たしたと、ページが実行される。セクション360では、「bind」要素は、メッセージ「/Call_connected」を受け取られたときに、「welcome」プロンプトを実行または再生し、「reco_drink」オブジェクトを実行することによって認識を開始する。音声サーバ204から受け取る認識結果と同様に、受け取るメッセージも多種多様である。メッセージの中には、所望のプログラムフローを開始するために言尾より定義されたものもある。その他のものも受け取って処理する(たとえば、受け取った認識サーバの認識結果と同様に解析する)ことができる。たとえばこれによ

って、キーボードからのテキストの本来の書体構文解析装置と同様に、マークアップを使用することができる。付録Aのreco要素は、この機能を実行するためのプロパティを含む。同様に、付録Aで詳細に説明するが、プロパティ「inner_text」を使用することによって、プロンプト要素を使用して動的コンテンツに可聴版ファイルにテキストメッセージを提供することができ、イベントインテグレーションに関するイベントインテグレーションと同等とすることができる。たとえば、イベントインテグレーションは「onReceived」を含むことが可能であり、これは、メッセージソース(たとえばクライアントデバイス上で実行中のアプリケーション)がブラウザに使用可能なメッセージを有する場合に送られる。【0087】したがって「smex」または「smex」オブジェクトは、本明細書で論じるマークアップタグを、クライアントデバイス上で実行中の他の構成要素またはアプリケーションまで拡張できるようにするものである。他の例として、メッセージオブジェクトは、クライアントデバイス上で実行中の耳の自由人向けTTY構成要素への通信に使用することができる。TTY構成要素は、音声認識を使用する代わりに、ユーザが入力した内容のメッセージを提供する。その後このメッセージは、認識サーバから認識結果が受け取られた場合と同じように割り当てられるか、または「bind」などのフィールドに使用されるか、または前述の「reco」要素を使用して、他の処理を実行することができる。メッセージまたは「smex」オブジェクトについては、付録Aで詳細に説明する。

【0088】「bind」要素は、「for」属性を含むことも可能であり、これによってその動作をページ上の他のオブジェクトに結びつけることができる。たとえば、以下のマークアップ

```
<bind for = "prompt 1" event = "onComplete"
  targetElement = "prompt 2" startMethod =
  "start" />
```

は、オブジェクト「prompt 1」がイベント「onComplete」を送信したときに、オブジェクト「prompt 2」のstartメソッドを呼び出す。

【0089】再度図5を参照すると、Webサーバ202はサーバ側のプラグイン宣言型ユーザインタフェースまたはモジュール320を含むことができる(たとえば、Microsoft CorporationによるASPまたはASP+、JSPなど)。サーバ側プラグインモジュール320は、クライアント側マークアップ、ならびにWebサーバ202にアクセス中のクライアントのタイプに特有な書式のマークアップでさえも、動的に生成することが可能である。クライアントサーバの関係が最初に確立された時点で、クライアント情報がW

33
Name属性を使用する場合と同様)。指定されない場

合、デフォルトの値は「value」。

・test: 任意選択。認識結果が割り当てられる条件を示すXML Pattern (W3C XML DOM指定の場合と同様) 文字列。デフォルト条件はtrue。

・value: 必須。ターゲット要素に割り当てられる認識結果ドキュメントからの値を指定する。XPath (W3C XML DOM指定の場合と同様) 文字列。
[0107] 例: 上記SMLの戻りが与えられると、以下のreco要素は、origin_cilityおよびdestination_cility内の値を、ターゲットページ要素textBoxOriginおよびtextBoxDestに転送するためにbindを使用する。

```
(input name="textBoxOrigin" type="text")
(input name="textBoxDest" type="text")

(reco id="travel")
(grammar src="/city.xml")
(bind targetElement="textBoxOrigin"
value="/origin_cility")
(bind targetElement="textBoxDest"
value="/dest_cility")
(/reco)
```

【0108】このbindは、以下の例に示されるよう*

```
<bind targetElement="textBoxDest"
value="/dest_city"
test="/xml/dest_cility(&confidence >= 40)"
/>
```

30
・babbleTimeout: 任意選択。認識装置が音声検出後に結果を戻さなければならないミリ秒単位の時間枠。automaticおよびsingingモードでのrecoの場合、これは音声検出から停止コールまでの時間枠に適用される。「multiple」モードでのrecoの場合、このタイムアウトは音声検出から各認識戻りまでの時間枠に適用され、すなわちこの時間枠は、それぞれの結果が戻されるかまたは他のイベント後に再開される。これを超えると、エラーが発生したか否かに従って、様々なイベントが投げられる。認識装置が依然として音声処理している場合、たとえば非常に長い発話の場合などは、statusコード13 (2、4、4を参照) でonNoRecoイベントが投げられる。ただし他のなんらかの理由でタイムアウトを超えた場合は、認識装置のエラーである可能性があるため、onTimeoutイベントが投げられる。指定されない場合は、音声プラットフォームがデフォルトとして内部値をとる。

34
・maxTimeout: 任意選択。認識開始から音声検出までのミリ秒単位の時間。この値が認識プラットフォームに渡され、これを超えると、認識プラットフォームからonSilenceイベントが提供される (2、4、2を参照)。指定されない場合、音声プラットフォームはデフォルト値を使用する。

35
・initialTimeout: 任意選択。認識開始から音声検出までのミリ秒単位の時間。この値が認識プラットフォームに渡され、これを超えると、認識プラットフォームからonSilenceイベントが提供される (2、4、2を参照)。指定されない場合、音声プラットフォームはデフォルト値を使用する。

36
・recoResult: 任意選択。認識プロセスを終了させるためのコールである。Recoオブジェクトは音声の録音を停止し、認識装置は受け取った音声に関する認識結果を、録音が停止した時点まで残す。Recoが使用するすべての認識リソースが解放され、その文法が非活動化される。 (このメソッドは、認識装置それ自身が、完全な文

37
・text: 録取り/復読み。認識された言葉のテキスト

トを保持する文字列 (すなわち、録取りモードでのrecoResultで、SML認識戻りでの最高位要素のテキスト属性のコンテンツに関する略記)。復読みモードの場合、文字列を割り当てることが可能であり、次いでこれが、文字列が認識結果に対応するのようにより解析される。復読みモードは、マークアップタグの拡張およびその処理を、クライアントデバイス上の他の構成要素またはアプリケーションに対して可能にするものである。文字列は「smex」メッセージオブジェクトから取得することができる。

38
・status: 録取り専用。認識プラットフォームから戻される状態コード。可能な値は、認識が成功した場合に0、または失敗値は-1から-4 (Starletメソッド (セクション2、3、1) およびActivateメソッド (セクション2、3、1) での可能な例外に定義されたとおり)、ならびに認識装置イベントの受取り時に設定される状態値-11から-15 (2、4を参照) である。

39
[0113] 2、3 オブジェクトメソッド
Recoの活動化およびgrammarの活動化は、RecoのDOMオブジェクトにあるいかなるメソッドを使用して制御することができる。これらのメソッドを使用すると、アップレベルブラウザがRecoオブジェクトを開始および停止すること、進行中の認識を取り消すこと、ならびに個々の文法最高位規則 (アップレベルブラウザのみ) を活動化および非活動化することができる。

40
[0114] 2、3、1 Start
Starletメソッドは、明示的に非活動化されていない認識文脈に対してすべての最高位規則を活動状態の文法として使用することで、認識プロセスを開始する。

41
[0115] 構文:
Object. Start ()
戻り値:
なし

42
例外: このメソッドは、非ゼロのstatusコードを設定し、録音時にはonNoRecoイベントを開始 (fire) する。可能な録音には、文法がない (reco status=-1)、文法のコンパイル障害と同様に様々な理由で発生することのある文法のロード障害、URIが存在しない (reco status=-2)、または音声プラットフォームのエラー (reco status=-3) が含まれる。

43
[0116] 2、3、2 Stop
Stopメソッドは、認識プロセスを終了させるためのコールである。Recoオブジェクトは音声の録音を停止し、認識装置は受け取った音声に関する認識結果を、録音が停止した時点まで残す。Recoが使用するすべての認識リソースが解放され、その文法が非活動化される。 (このメソッドは、認識装置それ自身が、完全な文

44
・recoResult: 録取り専用。2、1、2で記載したように、意味マークアップ言語 (SML) を含むXML DOMノードオブジェクト内に保持される認識結果。何も認識されない場合、プロパティはnullを戻す。

45
・text: 録取り/復読み。認識された言葉のテキスト

41	ハブル
抽出方法	なし 録音が停止する前に <code>maximize</code> 属性によって設定された 間隔が満了となった場合、ブラウザによって強いら れる。
デフォルト動作	<code>recStatus</code> を 1 に設定

【0143】 イベントプロパティ: イベントハンドラは * 各のハンドラである。発生する可能性のある様々なケ
ンプロパティを直接受け取らないが、ハンドラはデータに
関するイベントオブジェクトを照会することができる。

【0144】 2. 4. 4 `onNoReco: onNo` 【0145】 略文:

`Reco` は音声認識プラットフォームによって受けられ

たイベントが、有効な認識結果を戻すことができる * 【表 7】

インライン HTML	<code><script onNoReco="handler"></code>
イベントプロパティ	<code>Object.onNoReco = handler;</code> <code>Object.onNoReco =</code> <code>(DefaultHandler);</code>

※ 【表 8】

【0147】 イベントオブジェクト情報:

【0148】

41	ハブル
抽出方法	なし 録音状態が音声を検出したが、発話を検出することが できない
デフォルト動作	<code>status</code> プロパティを設定し、 <code>onNoReco</code> 属性を <code>status</code> コードは以下のように設定される。 <code>status-13</code> : 音声は検出されたが外音が続く可能性がある。 <code>status-14</code> : 何らかの音声が出たが検出されたが、 発話が十分に検出されなかった (しきい値設定については 2.2.1 の reject 属性を参照)。 <code>status-15</code> : 音声が出たが検出されたが、音声の検出 と <code>bubbleTimeout</code> 属性に指定された持続時間との間に 完全な認識を戻すことができなかった (2.2.1 を参 照)。

【0149】 イベントプロパティ: イベントハンドラは 力用のリソースを含む。単純な `prompt` は、以下の
プロパティを直接受け取らないが、ハンドラはデータに
関するイベントオブジェクトを照会することができる。

【0150】 3 `prompt`

`prompt` 要素は、システム出力を指定するのに使用
される。そのコンテンツは、以下の 1 つまたは複数であ
ってよい。

・インラインまたは参照済みテキスト。即時的またはそ
の他の音声出力情報でマークアップすることができる。
・含む間のドキュメントから明示的に取り出された要素

値
・音声ファイルへのリンク
【0151】 `prompt` 要素は、ダウンスケールブラウ
ザによって、またはアップレベルブラウザのオブジェク
トメンタによって、重層的に解釈される (または SM
IL コマンドによって活動化される)。

【0152】 3. 1 `prompt` のコンテンツ
`prompt` 要素は、テキストまたは音声ファイルの参
照のいずれか、あるいはその両方としての、システム出

50 `</prompt>`

【0154】 3. 1. 2 動的コンテンツ
`prompt` の実際のコンテンツは、`prompt` が出
力される直前にクライアント側で計算する必要がある場
合がある。特定の値を認識するために、たとえば整数内
値をデリファレンスする必要がある。その場合は、`v`
`alue` 要素を使用することができる。

【0155】 `Value` 要素
`value`: 任意選択。ドキュメント内にある要素の値
を取り出す。

10

行く先は

`targetAttribute="value" /`

から

`targetAttribute="value" /`

ですか。

`</prompt>`

【0158】 3. 1. 3 音声ファイル

`value` 要素は、合成された `prompt` の代わり

に、あるいはその中で再生するために、事前に録音され

た音声ファイルを参照するために使用することもできる。

以下の例では、`prompt` の終わりにピープという音

を鳴らす。

50

ビープ音の後にメッセージを録音してください。

`(value href="/wav/beep.wav" /)`

`</prompt>`

【0159】 3. 1. 4 参照済みプロンプト

コンテンツをインラインで指定する代わりに、空の要素

を備えた `src` 属性を使用することで、以下のように、

URI を介して外部コンテンツを参照することができ

る。

`<prompt id="Welcome" src="/ACMEWeatherPrompts/Welcome"/>`

`</prompt>`

`src` 属性のターゲットは、インラインプロンプト用に

指定された上記コンテンツのいずれかまたはすべてを保

持することができる。

【0160】 3. 2 属性およびプロパティ

`prompt` 要素は、以下の属性 (ダウンスケールブラウ

ザ) およびプロパティ (ダウンスケールおよびアップレベ

ルブラウザ) を保持する。

【0161】 3. 2. 1 属性

`title`: 任意選択。テキストから音声への合成に關す

るマークアップ言語タイプ。デフォルトの値は「SAP

1 51」。

`src`: インラインプロンプトが指定された場合は生

成選択。参照済みプロンプトの URI (3. 1. 4 を参

照)。

`bargain`: 任意選択。整数。プロンプトの開始

50

*場合、`targetAttribute` 属性を使用し
て、`targetElement` に必要な属性を指定す
ることができる。これは、たとえば HTML 形式印刷の
値をデリファレンスするために、`textBoxOrI`
`prompt` が出力される前に、`textBoxDest` 要素の
`value` 要素および `textBoxDest` 要素の
`prompt id="Confirm"`
行く先は
`(value targetElement="textBoxOrI")`
から
`(value targetElement="textBoxDest")`
targetAttribute="value" /

【0158】 3. 1. 3 音声ファイル

`value` 要素は、合成された `prompt` の代わり

に、あるいはその中で再生するために、事前に録音され

た音声ファイルを参照するために使用することもできる。

以下の例では、`prompt` の終わりにピープという音

を鳴らす。

50

ビープ音の後にメッセージを録音してください。

`(value href="/wav/beep.wav" /)`

`</prompt>`

【0159】 3. 1. 4 参照済みプロンプト

コンテンツをインラインで指定する代わりに、空の要素

を備えた `src` 属性を使用することで、以下のように、

URI を介して外部コンテンツを参照することができ

る。

`<prompt id="Welcome" src="/ACMEWeatherPrompts/Welcome"/>`

`</prompt>`

`src` 属性のターゲットは、インラインプロンプト用に

指定された上記コンテンツのいずれかまたはすべてを保

持することができる。

【0160】 3. 2 属性およびプロパティ

`prompt` 要素は、以下の属性 (ダウンスケールブラウ

ザ) およびプロパティ (ダウンスケールおよびアップレベ

ルブラウザ) を保持する。

【0161】 3. 2. 1 属性

`title`: 任意選択。テキストから音声への合成に關す

るマークアップ言語タイプ。デフォルトの値は「SAP

1 51」。

`src`: インラインプロンプトが指定された場合は生

成選択。参照済みプロンプトの URI (3. 1. 4 を参

照)。

`bargain`: 任意選択。整数。プロンプトの開始

50

は、promptのテキスト表現形式 (transcription) を提供し、これが合成装置に送られる。たとえば、prompt が可聴波ファイルの再生を含む場合、このプロパティはそのpromptのテキストバージョン (可聴波ファイルと共に格納されている場合が多い) を提供し、次いでたとえば、promptのテキストバージョンを構成要素またはクライアントデバイス上で実行中のアプリケーションに提供することによって、表示されるか、そうでなければ使用される。innerHTMLプロパティは、動的コンテンツを含むpromptのテキストバージョンを提供する際にも使用される。

[0163] 3. 3 promptメソッド
prompt再生は、promptのDOMオブジェクトにある以下のメソッドを使用して制御することができる。この方法では、アップレベルブラウザが、promptオブジェクトの開始 (start) および停止 (stop)、進行中のpromptの一時停止 (pause) および再開 (resume)、ならびに合成された音声の速度および音量の変更 (change) を実行することができる。

[0164] 3. 3. 1 Start
promptの再生を開始する。引数が与えられない限り、メソッドはオブジェクトのコンテンツを再生する。単一のpromptオブジェクトのみが所与の時間に「開始された」とみなされるため、Startが連続して呼び出されると、すべての再生が連続して行われる。

[0165] 構文:
Object. Start ([strText]);
パラメータ:
・strText: 合成装置に送信されるテキスト。存在する場合は、この引数がオブジェクトのコンテンツに優先される。
戻り値:
なし
例外:
・status=-1に設定すると、音声バンプがサーバーによって削除されている場合はonCompleteイベントが開始される。
[0166] 3. 3. 2 Pause
音声バンプをフラッシングせずに再生を一時停止する。再生が一時停止または停止されている場合、このメソッドは無効である。
[0167] 構文:
Object. Pause ();
戻り値:
なし

例外:
なし
[0168] 3. 3. 3 Resume
音声バンプをフラッシングせずに再生を再開する。再生が一時停止されていない場合、このメソッドは無効である。
[0169] 構文:
Object. Resume ();
戻り値:
なし
例外:
再生が失敗すると例外が投げられる。
[0170] 3. 3. 4 Stop
すでに停止されていない再生を停止し、音声バンプをフラッシングする。再生がすでに停止されている場合、このメソッドは音声バンプをフラッシングするだけである。
[0171] 構文:
Object. Stop ();
戻り値:
なし
例外:
なし

パラメータ:
・speed: 必須。変更する因数、speed=2. 0は現在の速度を2倍にする、speed=0. 5は現在の速度を半分にする、speed=0はデフォルト値を復元するという意味である。
・volume: 必須。変更する因数、volume=2. 0は現在の音量を2倍にする、volume=0. 5は現在の音量を半分にする、volume=0はデフォルト値を復元するという意味である。
戻り値:
なし
例外:
なし
[0174] 3. 3. 6 promptの制御例
以下の例は、上記のメソッドを使用して、キーワード読み機構をサポートしていないプラットフォームに対して、promptの制御をどのようにオーサリングできるかを示すものである。

```
<html>
<title> プロンプト制御 </title>
<head>
<script>
```

47 <!--
48 function checkKBargain() {
news.change(1.0, 0.5); // 録音中ボリュームを下げる。
if (keyword.text == "") { // 結果は閾値未満である。
news.change(1.0, 2.0); // ボリュームを元に戻す。
keyword.Start(); // 記録を再開する。
} else {
news.Stop(); // キーワードが検出された、プロンプトを停止する。
// 必要なことを行う。
}
//
</script>
<script for="window" event="onload">
<!--
news.Start(); keyword.Start();
//
</script>
</head>
<body>
<prompt id="news" bargain="0">
[0175] 水曜日、来週の結婚準備会を前に、投資家が何らかの大きな動きに出るにはほとんど刺激を受けなかったため、株は新たなパフォーマンスの低迷に入った。ハイテク株比率の高いナスダック総合指数は42. *
- <!--
</prompt>
</body>
<!--
reject="70"
onReco="checkKBargain()">
<grammar src="http://denali/news_bargain_grammar.xml" />
</reco>
</body>
</html>
[0176] 3. 4 Promptイベント
prompt DOMオブジェクトは以下のイベントをサポートしており、そのハンドラは、prompt変数の属性として指定することができる。
[0177] 3. 4. 1 onBookmark ※40 [表9]
[0178] 構文:
[0179]
※合成ブックマークに通過すると開始される。このイベントは再生を一時停止しない。
[0180] イベントオブジェクト情報:
[0181]

インラインHTML	Script onBookmark="handler">
イベントプロパティ	Object onbookmark = handler
	Object onbookmark =
	GetReco("handler")

バブル	なし
呼び出し方法	指定された文字列がブックマークに通過する
デフォルト動作	ブックマーク文字列を探索する

【0182】 イベントプロパティ：イベントハンドラはプロパティを直接受け取らないが、ハンドラはデータに関するイベントオブジェクトを照会することができる。
 【0183】 3. 4. 2 onBargain：ユーザのbargainイベントが検出されると開始される。(bargainイベントの構成内容、たとえば、
 【表11】

インラインHTML イベントプロパティ	Script onBargain="handler"> Object onBargain = handler Object onBargain = GetRef("handler")
------------------------	--

※ 【表12】

【0185】 イベントオブジェクト情報：

【0186】

バブル	なし
押し出し方法	bargain イベントに通過する
デフォルト動作	なし

【0187】 イベントプロパティ：イベントハンドラは★定義）に通過すると、開始される。

プロパティを直接受け取らないが、ハンドラはデータに関するイベントオブジェクトを照会することができる。
 【0189】 構文：
 【0190】
 【表13】
 【0188】 3. 4. 3 onComplete：prompt再生が終わりに達するか、または例外（上記に★20

インラインHTML イベントプロパティ	Script onComplete="handler"> Object onComplete = handler Object onComplete = GetRef("handler")
------------------------	---

☆ 【表14】

【0191】 イベントオブジェクト情報：

【0192】

バブル	なし
押し出し方法	prompt再生が完了する
デフォルト動作	再生が通常通りに完了した場合 status=0 に設定するが、そうでなければ status を上記で指定されたように設定する。

【0193】 イベントプロパティ：イベントハンドラはプロパティを直接受け取らないが、ハンドラはデータに関するイベントオブジェクトを照会することができる。
 【0194】 3. 4. 4 プックマークおよびイベントの使用
 以下の例は、prompt出力中にbargainが発生した場合に、ユーザの応答の意味、すなわち出発都市の修正または出発都市の準備のいずれであるかを特定するために、ブックマークイベントをどのように使用するかを示す例である。onBargainハンドラが、グローバル「mark」変数をprompt内で通過した最新のブックマークに設定するスクリプトを呼び出し、この「mark」の値が正しい値を設定するためにrecoの後処理関数（「heard」）で使われる。

```

var mark;
<script><![CDATA[
function interrupt() {
    mark = event.srcElement.bookmark;
}
function ProcessCityConfirm() {
    confirm.stop(); // オーディオバックアッファをフラッシュする。
    if (mark == "mark_origin_city")
        txtBoxOrigin.value =
event.srcElement.text;
    else
    ]]></script>

```

(27)

51

```

txtBoxDest.value =
event.srcElement.text;
}
</script>
<body>
<input name="txtBoxOrigin" value="シアトル"
type="text"/>
<input name="txtBoxDest" type="text" />
...
<prompt id="confirm" onBargain="interrupt0">
bargain="0">
From <bookmark mark="mark_origin_city" />
<value targetElement="origin"
targetAttribute="value" />.
<bookmark mark="mark_dest_city" /> が行く先であることを復唱し
てください。
</prompt>
<reco onReco="ProcessCityConfirm0">
<grammar src="/grm/1033/cities.xml" />
</reco>
...
</body>

```

【0195】 4 DTMF
DTMF認識オブジェクトを作成する。このオブジェクトはインラインマークアップ構文を使用する。DTMF認識オブジェクトはインスタンズ化することができる。DTMFは活動化されたときに、promptオブジェクトにbargainイベントを開始させることができ、DTMF認識に関連して以下で論じるタグおよびイベントに、音声ブラウザ216とメディアサーバ214との間に、音声ブラウザ216とメディアサーバ214との間の対話に關係するものである。
 【0196】 4. 1 コンテンツ
- dtmfgrammar：インライン文法用である。
例1：キーをテキストにマッピングする
 <input type="text" name="city" />
 <DTMF id="city_choice" timeout="2000" numDigits="1">
 <dtmfgrammar>
 <key value="1"> シアトル </key>
 <key value="2"> ボストン </key>
 </dtmfgrammar>
 <bind targetElement="city" targetAttribute="value" />
 </DTMF>

city_choice」が活動化されたときに、ユーザが1を押すと「Seattle」が、2を押すと「Boston」が入力フィールドに割り当てられ、それ以外の場合は何も割り当てられない。
 【0198】

city_choice」が活動化されたときに、ユーザが1を押すと「Seattle」が、2を押すと「Boston」が入力フィールドに割り当てられ、それ以外の場合は何も割り当てられない。
 【0198】

止する。

【0221】 構文:

* 【0222】
* 【第19】

インラインHTML	<DTMF onTimeout="handler">
イベントプロパティ (EXMASyntaxの場合)	Object onTimeout = handler Object onTimeout = GetDef("handler")

【0223】 イベントオブジェクト情報:

※ 【表20】

【0224】

バブル	なし
呼び出し方法	指定されたタイムアウト内に、何のDTMFサービスも ークも輸出されない
デフォルト動作	なし

【0225】 イベントプロパティ: イベントハンドラは

プロパティを直接受け取らないが、ハンドラはデータに

関するイベントオブジェクトを照会することができる。

【0226】 5 Callcontrolオブジェクト
電話音声ブラウザの電話インターフェース (呼、始末、
および接続) を表す。このオブジェクトは、GUIブラ
ウザでの windowオブジェクトと同じように固有で
ある。したがって、電話オブジェクトの存在期間はブラ
ウザインスタンスと自己と同一である。電話用の音声
ブラウザは、呼ごとに1つずつ電話オブジェクトイン
スタンス化する。ユーザはオブジェクトのインスタンス
化または配置は行わない。

【0227】 ここでは、このオブジェクトを介して第1
の当事者の呼制御に関する特徴だけを示す。

【0228】 5.1 プロパティ

・address
状態により異なる。発呼者のアドレスである。PST
Nの場合は、ANIとALIの組合せであってよい。V
oIPの場合は、これが発呼者のIPアドレスである。
・ringsBeforeAnswer 着信呼に回答する
前の呼出し音の回数。デフォルト値は無限であり、これ
は、開発者が呼に回答するために、具体的には以下のA
nswer () メソッドを使用しなければならぬこと
を意味する。電話局が着信呼を待ち行列に入れるのにA
CDを使用している場合、この番号は0に設定すること
ができる。

【0229】 5.2 メソッド

注: ここに示すメソッドはすべて同時に発生するもので

ある。

【0230】 5.2.1 Transfer

呼を転送する。ブライント転送の場合、転送が完了する

と、システムはオリジナルの呼およびフリーシステムリ

ソースを終了することができる。

構文:

telephone.Transfer (strText

);

パラメータ:

【0231】 5.2.2 Hangup

呼を終了する。

構文:

telephone.Hangup ();

パラメータ:

【0232】 5.2.3 Answer

呼に回答する。

構文:

telephone.Answer ()

戻り値:

なし

例外: 接続がない場合に例外が投げられる。この場合、

onAnswerイベントは開始されない。

【0234】 5.2.4 Hangup

呼を終了する。

構文:

telephone.Hangup ();

パラメータ:

呼を終了させる。現在進行中の呼がない場合は無効であ
る。

構文:

telephone.Hangup ()

戻り値:

なし

例外:

なし

【0235】 5.2.5 Connect

第1当事者の外に向かう呼を開始する。

構文:

telephone.Connect (strText

t, [iTimeout]);

パラメータ:

・strText: 必須。所期の受信者のアドレス。

・iTimeout: 任意選択。実行を中止するまでの

ミリ秒単位の時間。

指定されない場合、デフォルトの値はプラットフォーム

内部値である。

戻り値:

なし

例外: 途中番号に到達するか、あるいはFAXまたは留

守番電話に到達した場合を含み、呼が完了できない場合

に例外が投げられる (注: ハードウェアはこの機能をサ

ポートできない)。

【0236】 5.2.6 Record

ユーザの音声をファイルに記録する。

構文:

telephone.Record (url, ends

ilence, [maxTimeout], [init

ialTimeout]);

<HTML>

<HEAD>

<TITLE>Logon Page</TITLE>

</HEAD>

<SCRIPT>

var focus;

function RunSpeech ()

{

if (logon.user.value == "") {

focus="user";

p_uid.Start (); g_logon.Start (); dtmf.Start ();

return;

}

if (logon.pass.value == "") {

focus="pin";

p_pin.Start (); g_login.Start (); dtmf.Start ();

return;

}

p_thank.Start (); logon.submit ();

}

}

}

}

}

}

}

}

}

}

}

}

(32)

```

61 function login_reco0 {
    res = event.srcElement.recoResult;
    phlode = res.selectSingleNode("//uid");
    if (phlode != null)
        logon.user.value = phlode.xml;
    phlode = res.selectSingleNode("//password");
    if (phlode != null)
        logon.pass.value = phlode.xml;
}
function dtmf_reco0 {
    res = event.srcElement.dtmfResult;
    if (focus == "user")
        logon.user.value = res;
    else
        logon.pin.value = res;
}
</SCRIPT>
<SCRIPT for="callControl" event="onIncoming">
<!--
// 取り回しアドレス。カスタマイズされたものがあれば準備する。
callControl.Answer();
//
</SCRIPT>
<SCRIPT for="callControl" event="onOffhook">
<!--
    p_main.Start(); g_login.Start(); dtmf.Start();
    focus="user";
//
</SCRIPT>
<SCRIPT for="window" event="onload">
<!--
    if (logon.user.value != "") {
        p_retry.Start();
        logon.user.value = "";
        logon.pass.value = "";
        checkFields();
    }
//
</SCRIPT>
<BODY>
    <reco id="g_login"
        onReco="login_reco0; runSpeech0"
        timeout="5000"
        onTimeout="p_miss.Start(); RunSpeech0 >
        <grammar
            src=http://kaneel/stradedemo/speechonly/login.xml/>
        </reco >
        <dtmf id="dtmf"
            escape="#"
            onKeyPress="g_login.Stop();

```

```

63 onReco="dtmf_reco0;RunSpeech0"
interdigitTimeout="5000"
onTimeout="dtmf.Flush(); p_miss.Start();RunSpeech0" />

<prompt id="p_main"> ユーザIDおよびピン番号を復唱してください、</pro
mt>
<prompt id="p_uid"> ユーザIDだけ復唱してください、</prompt>
<prompt id="p_pin"> ピン番号だけ復唱してください、</prompt>
<prompt id="p_miss"> 申し訳ありません。失敗しました。</prompt>
<prompt id="p_thank"> ありがとうございます。身元を確認する間お待ちく
ださい、</prompt>
<prompt id="p_retry"> 申し訳ありません、ユーザIDおよびピン番号が一致
しません、</prompt>
<H2>Login</H2>
<form id="login">
    UID: <input name="user" type="text"
        onChange="runSpeech0" />
    PIN: <input name="pass" type="password"
        onChange="RunSpeech0" />
</form>
</BODY>
</HTML>
[0242] 6 ダイアログフローの制御
    ダイアログフローを実施するためのHTMLお
    ルブプロンプトのコンテンツ形成を助けるためにHTM
    上びスクリプトの使用の例は、入力ボックスに入れる
    値を探し、入力用の文脈依存ヘルプを提供する、単純な
    ザで使用する)で、title属性を使用する。
<html>
<title>Context Sensitive Help</title>
<head>
<script> var focus;
    function RunSpeech0 {
        if (trade.stock.value == "") {
            focus="trade.stock";
            p_stock.Start();
            return;
        }
        if (trade.op.value == "") {
            focus="trade.op";
            p_op.Start();
            return;
        }
        //... フィールドすべてについて上記を繰り返す。
        trade.submit();
    }
    function handle0 {
        res = event.srcElement.recoResult;
        if (res.text == "help") {
            text = "復唱してください";
            text += document.all[focus].title;
            p_help.Start(text);

```



```
69 <id>0903859304903498530985309094803</id>
    </param>
    </smex>
    <listen ...>
        ...// 入カフィールドの reco 結果を拘束する他の指示
        <bind targetElement="logServer" targetAttribute="sent"
            value="*[0log &#x3f; 3]" />
    </listen>
```

【0248】この例は、クラス id およびインタフェース id を備えた COM オブジェクトを使用して、ロギング機構をどのように構成するかを表現したものである。音声開発者は、ロギングの関心レベルを示す属性「log」を関連する SML ノードに付け加える。上記の例では、アプリケーション開発者は、単一の bind 指示文 (directive) を使用することによって、ログ値が 3 より大きいまたは等しいすべてのノードをログするように選択する。この例は、ダウンレベルブラウザとアップレベルブラウザの両方で使用できる。

【0249】この例は、smex オブジェクトが認識する

```
例 2: 着信呼のアドレスを読み取る
<input type="text" id="remote" />
<input type="text" id="transfer" />
<input type="text" id="local" />
<input type="hidden" id="session_id" />
```

```
...
<smex id="telephone" sent="startListening">
    <param name="server">http://tel-svr/whatever</param>
    <bind targetElement="session_id" value="//sid" />
    <bind targetElement="remote" value="//remote_addr" />
    <bind targetElement="transfer"
        value="//transfer_addr" />
    <bind targetElement="local" value="//local_addr" />
    ...
</smex>
```

【0251】この例は、受け取ったメッセージを処理するためにどのように bind 指示文を使用するかを示したものである。この例では、着信呼に使用するメッセージが、コンテキストがそれそれ着信呼のリモート、転送、およびローカルアドレスを表す、サブ要素 remote_addr、transfer_addr、および local_addr を有すると想定している。

【0252】この例では、電話サーバと通信するため、HTTP に基づく後続のないプロトコルが使用される。ここで電話サーバは、複数のブラウザインスタンスと通信するように設計されるため、各クライアントは、アプリケーションが開始されたときにサーバによって割り当てられた固有の id でそれ自体を識別しなければならぬ。これは、この例では、サーバに「startListening」メッセージを送信することによって達成されている。この例では、他の技法 (たとえばク

れる。プロパティが rvalue として使用される場合、またはこのプロパティに null オブジェクトが割り当てられる場合は無効である。

received: 読取り専用であって、受け取ったメッセージを示す XML DOM ノード。メッセージは、次の onReceive イベントの送信準備ができるまで、rvalue として使用可能となる。timer: 読取り/書き込みであって、タイムアウトイベントがトリガされるまでの時間を示す。ミリ秒単位の値。プロパティに正の値が割り当てられると、クロックは時間刻みを開始する。カウントダウンの進行中に、値を変更することができ。ゼロまたは負の値は、タイムアウトイベントをトリガせずにクロックを停止させる。デフォルトの値は 0 であり、タイムアウトがないことを意味する。

status: 読取り専用であって、オブジェクトの最近の状況を示す整数。可能な値は 0、-1、および 2 であって、それぞれ、正常、タイムアウト満了、ならびにブラットフォームとの通信が確立できないかまたは中断された、という意味である。ブラットフォーム特有のエラーメッセージは、received プロパティを介して搬送されなければならない。エラーメッセージが首尾よく送達された場合、status コードは 0 である。

【0254】7. 2 イベント

オブジェクトには以下のイベントがある。

```
<smex id="logServer" ...>... </smex>
<script> function logMessage(logClass, message) {
    logServer.sent = logClass + "|" + message;
} </script>
```

【0257】これは、実態には、動作がカスタマイズ可能な (グローバル) 関数を使用してオブジェクトを拡張するものである。上記の例では、ロギング関数は、id とメッセージとの間にフィールドセパレータ「|」を挿入するようにプログラムされる。

```
<smex id="logServer" onload="addFunction()" ... </smex>
<script>
    function my_logMessage(logClass, message) {
        logServer.sent = logClass + "|" + message;
    }
    function addFunction() {
        logServer.prototype.logMessage =
            my_logMessage;
    }
</script>
```

【0259】よりオブジェクト指向の強い方法での関数は、以下のようになすことができる。

```
logServer.logMessage(WC00_LOC_ERROR, "By
message");
```

【0260】すべての必要な機構はすでに標準で確立さ

* onReceive: このイベントは、ブラットフォームメッセージが到達したときに送信される。bind 要素によって宣言されたなんらかの指示文がある場合、それらの指示文は、イベントが開始される前に第 1 に評価される。送信の前に、received プロパティが更新される。

error: このイベントは、タイムアウトが発生したとき、または通信リンクエラーに遭遇したときに送信される。このイベントが送信されると、status プロパティは、前述のように対応するエラーコードで更新される。

【0255】7. 3 子要素

要素の形態を指定する場合、smex は以下の子要素を有することができる。

bind: 受け取ったメッセージ上で指示文が動作している場合を除き、reco の場合と同様。

param: reco の場合と同様に、smex オブジェクトにブラットフォーム特有のパラメータを提供する。各 param 要素には、「name」属性を使用して命名することが可能であり、param 要素のコンテンツがパラメータの値である。一実態形態では、要素は、名前スペースおよび XML データタイプ宣言用の標準 XML 属性を理解しなければならない。

【0256】7. 4 他の注釈

ロギング関数用に SMEX を拡張する確立方法の 1 つは、次のようになる。

```
<smex id="logServer" ...>... </smex>
<script> function logMessage(logClass, message) {
    logServer.sent = logClass + "|" + message;
} </script>
```

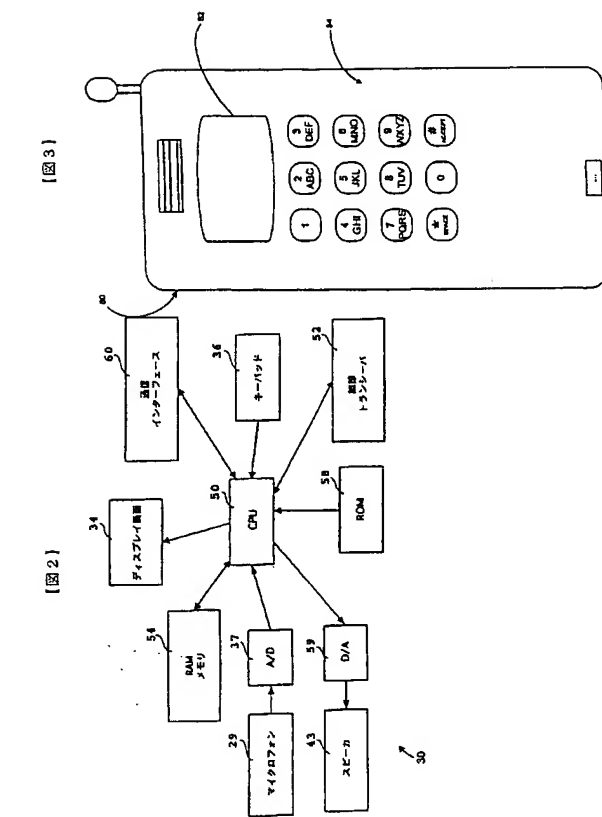
※ 【0258】グローバル関数を好まない人の場合、ECMAScript の「prototype」プロパティを使用して、オブジェクトメソッドとして関数を付け加えることができる。たとえば、以下のようになる。

```
<smex id="logServer" onload="addFunction()" ... </smex>
<script>
    function my_logMessage(logClass, message) {
        logServer.sent = logClass + "|" + message;
    }
    function addFunction() {
        logServer.prototype.logMessage =
            my_logMessage;
    }
</script>
```

※ ているが、上記の例に示すように、拡張作業を実行するための作業が smex オブジェクト実継者からさらに要求されることが留意された。

【図面の簡単な説明】

【図 1】コンピュータ・システム 50 の動作環境の第 1



【図 3】

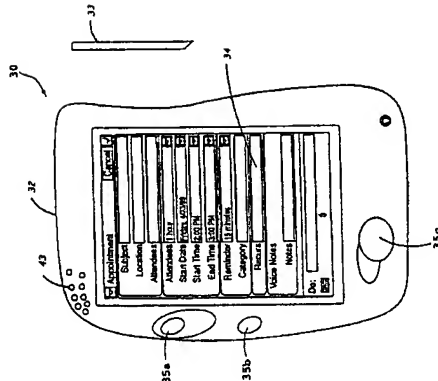
【図 2】

の実施形態を示す平面図である。
 【図 1】図 1 のコンピュータシステムの構成図である。
 【図 2】図 1 のコンピュータシステムの構成図である。
 【図 3】電話の平面図である。
 【図 4】汎用コンピュータの構成図である。
 【図 5】クライアント/サーバシステム用のアーキテクチャを示す構成図である。
 【図 6】クレジットカード情報を取得するための画面を示す図である。
 【図 7】クライアント側で実行可能なマークアップ言語のページを示す図である。
 【図 8】表示および音声認識機能を有する、クライアント側で実行可能なマークアップ言語のページを示す図である。
 【図 9】可聴表示専用および主導線がシステムにある、クライアント側で実行可能なマークアップ言語のページを示す図である。
 【図 10】可聴表示専用および主導線がシステムにある、クライアント側で実行可能なマークアップ言語のページを示す図である。
 【図 11】可聴表示専用および主導線がシステムにある、クライアント側で実行可能なマークアップ言語のページを示す図である。
 【図 12】可聴表示専用および主導線がシステムにある、クライアント側で実行可能なマークアップ言語のページを示す図である。
 【図 13】サーバ側プラグインモジュールによって実行可能なマークアップ言語のページを示す図である。

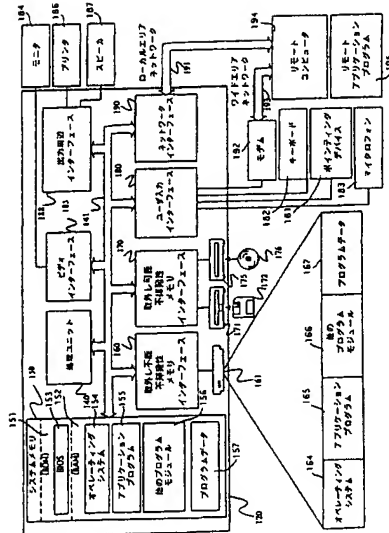
74

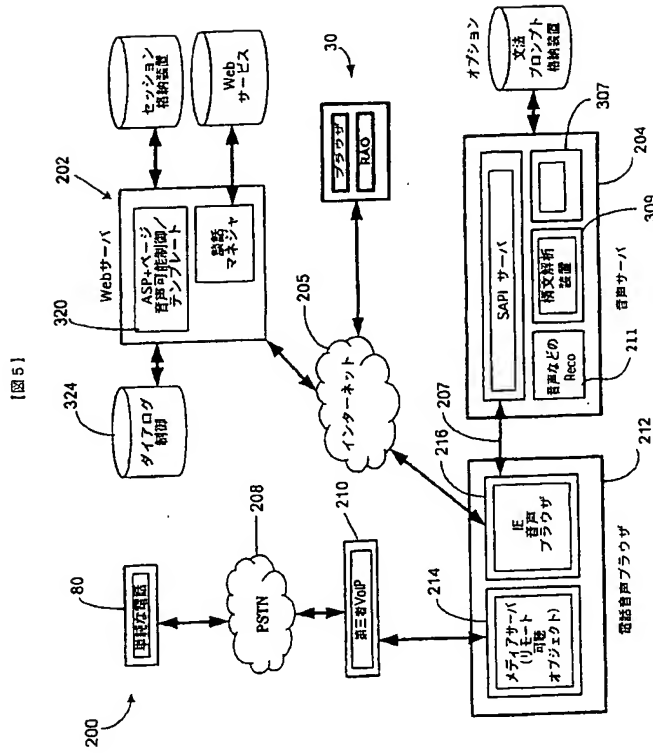
73

【図 1】

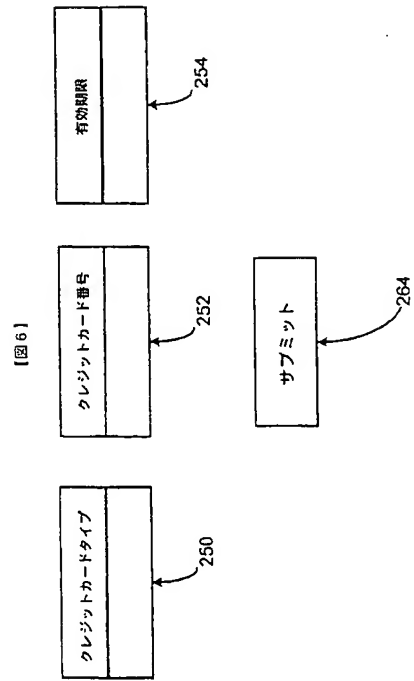


【図 4】





【图5】



【图6】

【圖 7】

```

<html>
  <form id="get_card_info" method="post" action="http://payment.asp">
    <select name="card_type">
      <option value="amex">American Express</option>
      <option value="visa">Visa</option>
      <option value="mc">MasterCard</option>
    </select>
    <input type="text" name="card_num" value="30"
      onChange="handle()" />
    <input type="text" name="expiry_data" />
    <input type="submit" value="Submit" onClick="verify()" />
  </form>
  <script>
    function handle() {
      if (get_card_info.card_type.value == "amex") {
        if (get_card_info.card_num.length != 16)
          alert ("amex should have 16 digits");
        else
          if (get_card_info.card_num.length != 16)
            alert ("visa and master should have 16 digits");
      }
      function verify() {
        var obj = document.getElementById("get_card_info");
        if (obj)
          obj.submit();
      }
    }
  </script>
</html>

```

[X10]

```

function _handleO {
    handleO();
    checkFilled();
}
function checkFilled() {
    if (card_type.value == "") {
        p_card_type.activate();
        return;
    }
    if (card_num.value == "") {
        p_card_num.activate();
        return;
    }
    if (expiry_date.value == "") {
        p_expiry_date.activate();
        return;
    }
    p_content.activate();
    p_confirm.activate();
    confirmation.activate();
}
function confirmed(gobj) {
    if (gobj.get_card_info.text == "yes")
        if (gobj.get_card_info.submit())
        {
            // user codes start here
            //
            //
            function handleO () {
                if (get_card_info.card_type == "anex") {
                    if (get_card_info.card_num.length != 15) {
                        prompt.speak ("anex should have 15 digits");
                        get_card_info.card_num = "";
                    }
                }
                else {
                    if (get_card_info.card_num.length != 16) {
                        prompt.speak ("visa and master should have 16 digits");
                        get_card_info.card_num = "";
                    }
                }
            }
        }
    }
}
</script>
</body>

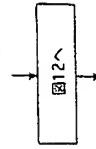
```


【図11】

```

<body>
  <form id="get_card_info" method="post" action="http://payment.asp"
  onactivate="welcome()">
    <prompt id="p_welcome">We now need your credit card</prompt>
    <prompt id="p_mumble">I didn't understand you</prompt>
    <prompt id="p_card_type" bargain="true">What credit card would you
    use?</prompt>
    <prompt id="p_card_num" bargain="true">Please say the number</prompt>
    <prompt id="p_expiry_date" bargain="true">What is the expiration
    date?</prompt>
    <prompt id="p_content">
      I have your <value select="card_type" /> <value select="card_num" />
      with expiration date <value select="expiry_date" />
    </prompt>
    <prompt id="p_confirm">Is this correct?</prompt>
    <reco id="g_get_card_info" onReco="handle()" onNoReco="mumble(this)">
    <grammar src = "/gram/getPayment" />
    <bind target="card_type" value="/card/type" />
    <bind target="card_num" value="/card/number" />
    <bind target="expiry_date" value="/card/expr_date" />
    </reco>
    <reco id="confirmation" onReco = "confirmed(this)" onNoReco="mumble(this)" />
    <grammar src = "/gram/yesno" />
    <select name="card_type">
    <option value="amex">American Express</option>
    <option value="visa">Visa</option>
    <option value="ms">MasterCard</option>
    </select>
    <input type="text" name="card_num" width="30" />
    <input type="text" name="expiry_date" />
    <input type="submit" value="Submit" />
    </form>
    <script><![CDATA[
      function welcome() {
        p_welcome.active();
        checkFilled();
      }
      function mumble(gobj) {
        gobj.deactivate();
        p_mumble.active();
        checkFilled();
      }
      function handle() {
        handleQ_filled();
        checkFilled();
      }
    ]]>
  </script>

```



【図12】

```

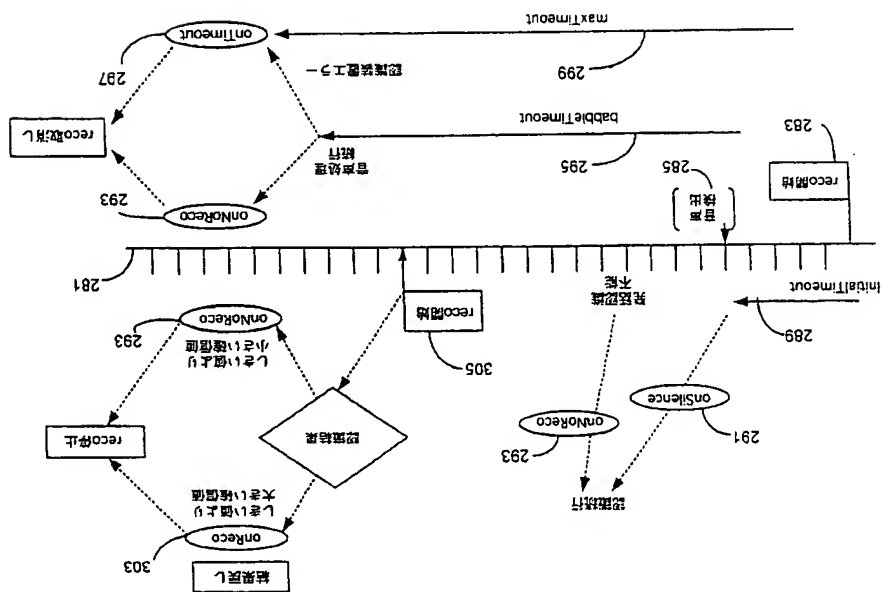
function checkFilled() {
  if (card_type.value == "") {
    p_card_type.active(); do_field.activate(); return;
  }
  if (card_num.value == "") {
    p_card_num.active(); do_field.activate(); return;
  }
  if (expiry_date.value == "") {
    p_expiry_date.active(); do_field.activate(); return;
  }
  p_content.activate();
  p_confirm.activate();
  confirmation.activate();
  function confirmed(gobj) {
    if (gobj.recognizes.text == "yes")
      get_card_info.submit(gensai());
  }
  // user codes start here
  //
  function handle() {
    if (field == get_card_info.card_num) {
      if (get_card_info.card_num.length != 15) {
        prompt.speak("Name should have 15 digits");
        get_card_info.card_num = "";
      }
    } else {
      if (get_card_info.card_num.length != 16) {
        prompt.speak("visa should have 15 digits");
        get_card_info.card_num = "";
      }
    }
  }
  function gensai() {
    str = "csm"><credit_card type="";
    str += card_type.value; str += "<numbers";
    str += card_number.value; str += "</numbers>";
    str += expiry_date.value; str += "</expiry_date>";
    return str;
  }
}
</script>
</body>

```


特開2003-131772

(48)

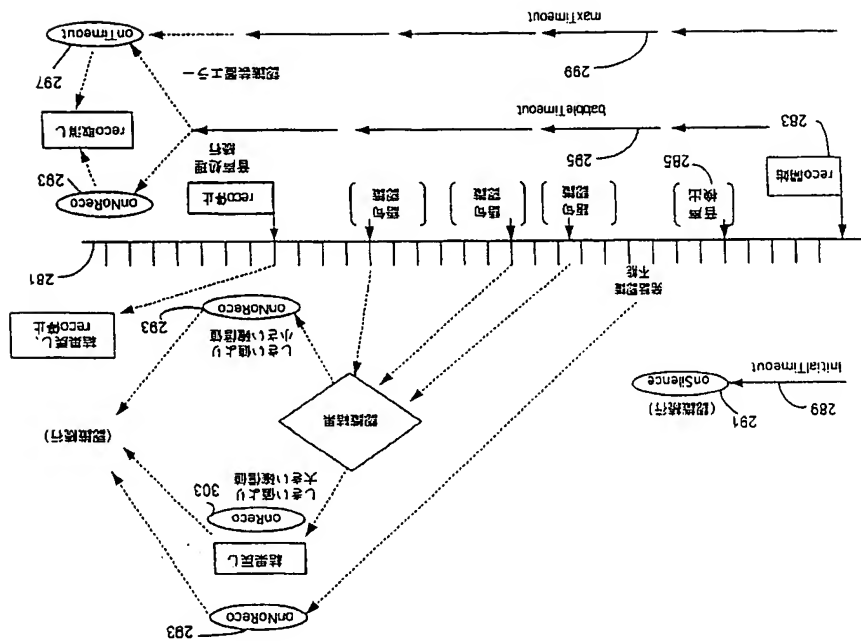
【圖 15】



特開2003-131772

(49)

【图 16】



【外国語明細書】

1. Title of the Invention

**MARKUP LANGUAGE EXTENSIONS FOR WEB ENABLED
RECOGNITION**

2. Claims

1. A computer readable medium having a markup language for execution on a client device in a client/server system, the markup language comprising an instruction indicating a grammar to associate with input entered through the client device.
2. The computer readable medium of claim 1 wherein the instruction indicating a grammar provides a reference to a location of the grammar.
3. The computer readable medium of claim 1 wherein the grammar is for one of speech recognition, handwriting recognition, gesture recognition and visual recognition.
4. The computer readable medium of claim 3 wherein the instruction indicating a grammar includes a reference to a grammar for speech recognition.
5. The computer readable medium of claim 1 wherein the markup language includes an instruction for receiving a recognition result pertaining to recognized speech and associating the result with a data field on the client device.
6. The computer readable medium of claim 8 wherein the instruction for receiving a recognition result associates the recognition result to a plurality of data fields.

7. The computer readable medium of claim 1 wherein the markup language includes an instruction to audibly render an indication when the speech was not recognized.
8. The computer readable medium of claim 1 wherein the input comprises a text string.
9. The computer readable medium of claim 8 wherein the markup language includes an instruction for receiving the text string from another component.
10. The computer readable medium of claim 9 wherein markup language associates the grammar to process the text string.
11. A computer readable medium including instructions readable by a computer which, when implemented, cause the computer to handle information by performing steps comprising:
 - receiving a markup language page from a web server having a field for input data;
 - receiving input from a user related to the field; and
 - sending data indicative of the input and an indication of a grammar for recognition.
12. The computer readable medium of claim 11 wherein the indication provides a reference to locations of the grammars.
13. The computer readable medium of claim 12 wherein the indication includes a reference to a language for speech recognition.

14. A method for recognition in a client/server network, the method comprising:

receiving a markup language page having a field for input data on a client device connected to a network from a web server at an address on the network different than the client device;
receiving input from a user related to the field on the client device; and
sending data indicative of the input and an indication of a grammar for recognition to a recognition server at an address on the network different than the client device.

15. The method of claim 14 and further comprising:
normalizing the data indicative of the input prior to sending the data.

16. The method of claim 41 and further comprising:
receiving a recognition result pertaining to recognized speech and associating the result with a data field on the client device.

17. A computer readable medium having a markup language for execution on a client device in a client/server system, the markup language comprising an instruction indicating an object model element having an attribute or a method for recognition.

18. The computer readable medium of claim 17 wherein the element is for one of speech recognition, handwriting recognition, gesture recognition, DTMF recognition and visual recognition.

19. The computer readable medium of claim 17 wherein the element comprises an indication of a grammar for recognition.

20. The computer readable medium of claim 19 wherein element pertains to receiving and processing a text string like recognition.

21. The computer readable medium of claim 20 and further comprising a second object model receiving the text string from another component.

22. A computer readable medium having a markup language for execution on a client device in a client/server system, the markup language comprising an instruction indicating an object model element having an attribute or a method for audibly prompting.

23. The computer readable medium of claim 22 wherein the markup language comprises one of HTML, XHTML, cHTML, XML and WML.

24. The computer readable medium of claim 22 wherein the markup language comprises a scripting language.

25. The computer readable medium of claim 22 wherein the markup language comprises a synchronized markup language.

3. Detailed Description of the Invention

FIELD OF THE INVENTION

The present invention relates to access of information over a wide area network such as the Internet. More particularly, the present invention relates to web enabled recognition allowing information and control on a client side to be entered using a variety of methods.

BACKGROUND OF THE INVENTION

Small computing devices such as personal information managers (PIM), devices and portable phones are used with ever increasing frequency by people in their day-to-day activities. With the increase in processing power now available for microprocessors used to run these devices, the functionality of these devices are increasing, and in some cases, merging. For instance, many portable phones now can be used to access and browse the Internet as well as can be used to store personal information such as addresses, phone numbers and the like.

In view that these computing devices are being used for browsing the Internet, or are used in other server/client architectures, it is therefore necessary to enlarge information into the computing device. Unfortunately, due to the desire to keep these devices as small as possible in order that they are easily carried, conventional keyboards having all the letters of the alphabet as isolated buttons are usually not possible due to the limited surface area available on the housings of the computing devices.

Recently, voice portals such as through the use of VoiceXML (voice extensible markup language) have been advanced to allow Internet content to be accessed using only a telephone. In this architecture, a document server (for

example, a web server) processes requests from a client through a VoiceXML interpreter. The web server can produce VoiceXML documents in reply, which are processed by the VoiceXML interpreter and rendered audibly to the user. Using voice commands through voice recognition, the user can navigate the web.

VoiceXML is a markup language with flow control tags; however, flow control does not follow the HTML (Hyper Text Markup Language) flow control model, which includes eventing and separate scripts. Rather, VoiceXML generally includes a form interpretation algorithm that is particularly suited for telephone-based voice-only interaction, and commonly, where the information obtained from the user is under the control of the system or application. Incorporation of VoiceXML directly into applications available in a client-server relationship where graphically user interfaces are also provided will require the developer to master two forms of web authoring, one for VoiceXML and the other using HTML (or the like), each one following a different flow control model.

There is thus an ongoing need to improve upon the architecture and methods used to provide speech recognition in a server/client architecture such as the Internet. The authoring tool for speech recognition should be easily adaptable to small computing devices such as PIMs, telephones and the like. An architecture or method of web authoring that addresses one, several or all of the foregoing disadvantages is particularly needed.

SUMMARY OF THE INVENTION

A markup language for execution on a client device in a client/server system includes an instruction indicating a grammar to associate with input data entered through the client device.

With the availability of this extension and as another aspect of the present invention, a client device can execute instructions to receive a markup language page from a web server having a field for input data. The client device can then receive input data from a user related to the field and send the data and an indication of the grammar for recognition to a recognition server, typically, located at a remote location for processing.

The recognition server can execute instructions to receive the input data and the indication of the grammar to perform recognition. The results of recognition can then be sent back to the client device or web server for further processing.

DETAILED DESCRIPTION OF THE INVENTION

Before describing an architecture of web based recognition and methods for implementing the same, it may be useful to describe generally computing devices that can function in the architecture. Referring now to FIG. 1, an exemplary form of a data management device (PIM, PDA or the like) is illustrated at 30. However, it is contemplated that the present invention can also be practiced using other computing devices discussed below, and in particular, those computing devices having limited surface areas for input buttons or the like. For example, phones and/or data management devices will also benefit from the present invention. Such devices will have an enhanced utility compared to existing portable personal information management devices and other portable electronic devices, and the functions and compact size of such devices will more likely encourage the user to carry the device at all times. Accordingly, it is not intended that the scope of the architecture herein described be limited by the disclosure of

an exemplary data management or PIM device, phone or computer herein illustrated.

An exemplary form of a data management mobile device 30 is illustrated in FIG. 1. The mobile device 30 includes a housing 32 and has an user interface including a display 34, which uses a contact sensitive display screen in conjunction with a stylus 33. The stylus 33 is used to press or contact the display 34 at designated coordinates to select a field, to selectively move a starting position of a cursor, or to otherwise provide command information such as through gestures or handwriting. Alternatively, or in addition, one or more buttons 35 can be included on the device 30 for navigation. In addition, other input mechanisms such as rotatable wheels, rollers or the like can also be provided. However, it should be noted that the invention is not intended to be limited by these forms of input mechanisms. For instance, another form of input can include a visual input such as through computer vision.

Referring now to FIG. 2, a block diagram illustrates the functional components comprising the mobile device 30. A central processing unit (CPU) 50 implements the software control functions. CPU 50 is coupled to display 34 so that text and graphic icons generated in accordance with the controlling software appear on the display 34. A speaker 43 can be coupled to CPU 50 typically with a digital-to-analog converter 59 to provide an audible output. Data that is downloaded or entered by the user into the mobile device 30 is stored in a non-volatile read/write random access memory store 54 bi-directionally coupled to the CPU 50. Random access memory (RAM) 54 provides volatile storage for instructions that are executed by CPU 50, and storage for temporary data, such as register values. Default values for

configuration options and other variables are stored in a read only memory (ROM) 58. ROM 58 can also be used to store the operating system software for the device that controls the basic functionality of the mobile 30 and other operating system kernel functions (e.g., the loading of software components into RAM 54).

RAM 54 also serves as a storage for the code in the manner analogous to the function of a hard drive on a PC that is used to store application programs. It should be noted that although non-volatile memory is used for storing the code, it alternatively can be stored in volatile memory that is not used for execution of the code.

Wireless signals can be transmitted/received by the mobile device through a wireless transceiver 52, which is coupled to CPU 50. An optional communication interface 60 can also be provided for downloading data directly from a computer (e.g., desktop computer), or from a wired network, if desired. Accordingly, interface 60 can comprise various forms of communication devices, for example, an infrared link, modem, a network card, or the like.

Mobile device 30 includes a microphone 29, and analog-to-digital (A/D) converter 37, and an optional recognition program (speech, DTMF, handwriting, gesture or computer vision) stored in store 54. By way of example, in response to audible information, instructions or commands from a user of device 30, microphone 29 provides speech signals, which are digitized by A/D converter 37. The speech recognition program can perform normalization and/or feature extraction functions on the digitized speech signals to obtain intermediate speech recognition results. Using wireless transceiver 52 or communication interface 60, speech data is transmitted to a remote recognition server 204 discussed below and illustrated

in the architecture of FIG. 5. Recognition results are then returned to mobile device 30 for rendering (e.g. visual and/or audible) thereon, and eventual transmission to a web server 202 (FIG. 5), wherein the web server 202 and mobile device 30 operate in a client/server relationship. Similar processing can be used for other forms of input. For example, handwriting input can be digitized with or without pre-processing on device 30. Like the speech data, this form of input can be transmitted to the recognition server 204 for recognition wherein the recognition results are returned to at least one of the device 30 and/or web server 202. Likewise, DTMF data, gesture data and visual data can be processed similarly. Depending on the form of input, device 30 (and the other forms of clients discussed below) would include necessary hardware such as a camera for visual input.

FIG. 3 is a plan view of an exemplary embodiment of a portable phone 80. The phone 80 includes a display 82 and a keypad 84. Generally, the block diagram of FIG. 2 applies to the phone of FIG. 3, although additional circuitry necessary to perform other functions may be required. For instance, a transceiver necessary to operate as a phone will be required for the embodiment of FIG. 2; however, such circuitry is not pertinent to the present invention.

In addition to the portable or mobile computing devices described above, it should also be understood that the present invention can be used with numerous other computing devices such as a general desktop computer. For instance, the present invention will allow a user with limited physical abilities to input or enter text into a computer or other computing device when other conventional input devices, such as a full alpha-numeric keyboard, are too difficult to operate.

The invention is also operational with numerous other general purpose or special purpose computing systems, environments or configurations. Examples of well known computing systems, environments, and/or configurations that may be suitable for use with the invention include, but are not limited to, regular telephones (without any screen) personal computers, server computers, hand-held or laptop devices, multiprocessor systems, microprocessor-based systems, set top boxes, programmable consumer electronics, network PCs, minicomputers, mainframe computers, distributed computing environments that include any of the above systems or devices, and the like.

The following is a brief description of a general purpose computer 120 illustrated in FIG. 4. However, the computer 120 is again only one example of a suitable computing environment and is not intended to suggest any limitation as to the scope of use or functionality of the invention. Neither should the computer 120 be interpreted as having any dependency or requirement relating to any one or combination of components illustrated therein.

The invention may be described in the general context of computer-executable instructions, such as program modules, being executed by a computer. Generally, program modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types. The invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote computer storage media including memory storage devices. Tasks performed by the

programs and modules are described below and with the aid of figures. Those skilled in the art can implement the description and figures as processor executable instructions, which can be written on any form of a computer readable medium.

With reference to FIG. 4, components of computer 120 may include, but are not limited to, a processing unit 140, a system memory 150, and a system bus 141 that couples various system components including the system memory to the processing unit 140. The system bus 141 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. By way of example, and not limitation, such architectures include Industry Standard Architecture (ISA) bus, Universal Serial Bus (USB), Micro Channel Architecture (MCA) bus, Enhanced ISA (EISA) bus, Video Electronics Standards Association (VESA) local bus, and Peripheral Component Interconnect (PCI) bus also known as Mezzanine bus. Computer 120 typically includes a variety of computer readable mediums. Computer readable mediums can be any available media that can be accessed by computer 120 and includes both volatile and nonvolatile media, removable and non-removable media. By way of example, and not limitation, computer readable mediums may comprise computer storage media and communication media. Computer storage media includes both volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information such as computer readable instructions, data structures, program modules or other data. Computer storage media includes, but is not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital

versatile disks (DVD) or other optical disk storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can be accessed by computer 120.

Communication media typically embodies computer readable instructions, data structures, program modules or other data in a modulated data signal such as a carrier wave or other transport mechanism and includes any information delivery media. The term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media includes wired media such as a wired network or direct-wired connection, and wireless media such as acoustic, FR, infrared and other wireless media. Combinations of any of the above should also be included within the scope of computer readable media.

The system memory 150 includes computer storage media in the form of volatile and/or nonvolatile memory such as read only memory (ROM) 151 and random access memory (RAM) 152. A basic input/output system 153 (BIOS), containing the basic routines that help to transfer information between elements within computer 120, such as during start-up, is typically stored in ROM 151. RAM 152 typically contains data and/or program modules that are immediately accessible to and/or presently being operated on by processing unit 140. By way of example, and not limitation, FIG. 4 illustrates operating system 54, application programs 155, other program modules 156, and program data 157.

The computer 120 may also include other removable/non-removable volatile/nonvolatile computer storage media. By way of example only, FIG. 4 illustrates a hard disk drive 161 that reads from or writes to non-removable, nonvolatile magnetic media, a magnetic disk drive 171 that reads from or writes to a removable, nonvolatile magnetic disk 172, and an optical disk drive 175 that reads from or writes to a removable, nonvolatile optical disk 176 such as a CD ROM or other optical media. Other removable/non-removable, volatile/nonvolatile computer storage media that can be used in the exemplary operating environment include, but are not limited to, magnetic tape cassettes, flash memory cards, digital versatile disks, digital video tape, solid state RAM, solid state ROM, and the like. The hard disk drive 161 is typically connected to the system bus 141 through a non-removable memory interface such as interface 160, and magnetic disk drive 171 and optical disk drive 175 are typically connected to the system bus 141 by a removable memory interface, such as interface 170.

The drives and their associated computer storage media discussed above and illustrated in FIG. 4, provide storage of computer readable instructions, data structures, program modules and other data for the computer 120. In FIG. 4, for example, hard disk drive 161 is illustrated as storing operating system 164, application programs 165, other program modules 166, and program data 167. Note that these components can either be the same as or different from operating system 154, application programs 155, other program modules 156, and program data 157. Operating system 164, application programs 165, other program modules 166,

and program data 167 are given different numbers here to illustrate that, at a minimum, they are different copies.

A user may enter commands and information into the computer 120 through input devices such as a keyboard 182, a microphone 183, and a pointing device 181, such as a mouse, trackball or touch pad. Other input devices (not shown) may include a joystick, game pad, satellite dish, scanner, or the like. These and other input devices are often connected to the processing unit 140 through a user input interface 180 that is coupled to the system bus, but may be connected by other interface and bus structures, such as a parallel port, game port or a universal serial bus (USB). A monitor 184 or other type of display device is also connected to the system bus 141 via an interface, such as a video interface 185. In addition to the monitor, computers may also include other peripheral output devices such as speakers 187 and printer 186, which may be connected through an output peripheral interface 188.

The computer 120 may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 134. The remote computer 134 may be a personal computer, a hand-held device, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to the computer 120. The logical connections depicted in FIG. 4 include a local area network (LAN) 191 and a wide area network (WAN) 193, but may also include other networks. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets and the Internet.

When used in a LAN networking environment, the computer 120 is connected to the LAN 191 through a network interface or adapter 190. When used in a WAN networking environment, the computer 120 typically includes a modem 192 or other means for establishing communications over the WAN 193, such as the Internet. The modem 192, which may be internal or external, may be connected to the system bus 141 via the user input interface 180, or other appropriate mechanism. In a networked environment, program modules depicted relative to the computer 120, or portions thereof, may be stored in the remote memory storage device. By way of example, and not limitation, FIG. 4 illustrates remote application programs 195 as residing on remote computer 194. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

FIG. 5 illustrates architecture 200 for web based recognition as can be embodied in the present invention. Generally, information stored in a web server 202 can be accessed through mobile device 30 (which herein also represents other forms of computing devices having a display screen, a microphone, a camera, a touch sensitive panel, etc., as required based on the form of input), or through phone 80 wherein information is requested audibly or through tones generated by phone 80 in response to keys depressed and wherein information from web server 202 is provided only audibly back to the user.

More importantly though, architecture 200 is unified in that whether information is obtained through device 30 or phone 80 using speech recognition, a single recognition server 204 can support either mode of operation. In addition, architecture 200 operates using an

extension of well-known mark-up languages (e.g. HTML, XHTML, cHTML, XML, WML, and the like). Thus, information stored on web server 202 can also be accessed using well-known GUI methods found in these mark-up languages. By using an extension of well-known mark-up languages, authoring on the web server 202 is easier, and legacy applications currently existing can be also easily modified to include voice recognition.

Generally, device 30 executes HTML pages, scripts, or the like, provided by web server 202. When voice recognition is required, by way of example, speech data, which can be digitized audio signals or speech features wherein the audio signals have been preprocessed by device 30 as discussed above, are provided to recognition server 204 with an indication of a grammar or language model to use during speech recognition. The implementation of the recognition server 204 can take many forms, one of which is illustrated, but generally includes a recognizer 211. The results of recognition are provided back to device 30 for local rendering if desired or appropriate. Upon compilation of information through recognition and any graphical user interface if used, device 30 sends the information to web server 202 for further processing and receipt of further HTML pages/scripts, if necessary.

As illustrated in FIG. 5, device 30, web server 202 and recognition server 204 are commonly connected, and separately addressable, through a network 205, herein a wide area network such as the Internet. It therefore is not necessary that any of these devices be physically located adjacent each other. In particular, it is not necessary that web server 202 includes recognition server 204. In

this manner, authoring at web server 202 can be focused on the application to which it is intended without the authors needing to know the intricacies of recognition server 204. Rather, recognition server 204 can be independently designed and connected to the network 205, and thereby, be updated and improved without further changes required at web server 202. As discussed below, web server 202 can also include an authoring mechanism that can dynamically generate client-side markups and scripts. In a further embodiment, the web server 202, recognition server 204 and client 30 may be combined depending on the capabilities of the implementing machines. For instance, if the client comprises a general purpose computer, e.g. a personal computer, the client may include the recognition server 204. Likewise, if desired, the web server 202 and recognition server 204 can be incorporated into a single machine.

An aspect of the present invention is a method for processing input data in a client/server system that includes receiving from a server a markup language page having extensions configured to obtain input data from a user of a client device; executing the markup language page on the client device; transmitting input data (indicative of speech, DTMF, handwriting, gestures or images obtained from the user) and an associated grammar to a recognition server remote from the client; and receiving a recognition result from the recognition server at the client. Another aspect is a computer-readable medium having a markup language for execution on a client device in a client/server system, the markup language having an instruction indicating a grammar to associate with input data entered through the client device.

Access to web server 202 through phone 80 includes connection of phone 80 to a wired or wireless telephone network 208, that in turn, connects phone 80 to a third party gateway 210. Gateway 210 connects phone 80 to a telephony voice browser 212. Telephony voice browser 212 includes a media server 214 that provides a telephony interface and a voice browser 216. Like device 30, telephony voice browser 212 receives HTML pages/scripts or the like from web server 202. More importantly though, the HTML pages/scripts are of the form similar to HTML pages/scripts provided to device 30. In this manner, web server 202 need not support device 30 and phone 80 separately, or even support standard GUI clients separately. Rather, a common mark-up language can be used. In addition, like device 30, voice recognition from audible signals transmitted by phone 80 are provided from voice browser 216 to recognition server 204, either through the network 205, or through a dedicated line 207, for example, using TCP/IP. Web server 202, recognition server 204 and telephone voice browser 212 can be embodied in any suitable computing environment such as the general purpose desktop computer illustrated in FIG. 4.

However, it should be noted that if DTMF recognition is employed, this form of recognition would generally be performed at the media server 214, rather than at the recognition server 204. In other words, the DTMF grammar would be used by the media server.

As indicated above, one aspect of the present invention includes extension of mark-up languages such as HTML, XHTML, CHTML, XML, WML or with any other SGML-derived markup to include controls and/or objects that provide recognition in a client/server architecture. In this

manner, authors can leverage all the tools and expertise in these mark-up languages that are the predominant web development platform used in such architectures.

Generally, controls and/or objects can include one or more of the following functions: recognizer controls and/or objects for recognizer configuration, recognizer execution and/or post-processing; synthesizer controls and/or objects for synthesizer configuration and prompt playing; grammar controls and/or objects for specifying input grammar resources; and/or binding controls and/or objects for processing recognition results. The extensions are designed to be a lightweight markup layer, which adds the power of an audible, visual, handwriting, etc. interface to existing markup languages. As such, the extensions can remain independent of: the high-level page in which they are contained, e.g. HTML; the low-level formats which the extensions used to refer to linguistic resources, e.g. the text-to-speech and grammar formats; and the individual properties of the recognition and speech synthesis platforms used in the recognition server 204.

Before describing mark-up languages having controls and/or objects suited for recognition, it may be helpful to examine a simple GUI example herein embodied, with the HTML mark-up language. Referring to FIG. 6, a simple GUI interface comprises submission of credit card information to the web server to complete an on-line sale. In this example, the credit card information includes a field 250 for entry of the type of credit card being used, for example, Visa, MasterCard or American Express. A second field 252 allows entry of the credit card number, while a third field 254 allows entry of the expiration date. Submit

button 264 is provided to transmit the information entered in fields 250, 252 and 254.

FIG. 7 illustrates the HTML code for obtaining the foregoing credit card information from the client. Generally, as is common in these forms of mark-up languages, the code includes a body portion 260 and a script portion 262. The body portion 260 includes lines of code indicating the type of action to be performed, the form to use, the various fields of information 250, 252 and 254, as well as a code for submit button 264 (FIG. 6). This example also illustrates eventing support and embedded script hosting, wherein upon activation of the submit button 264, a function "verify" is called or executed in script portion 262. The "verify" function ascertains whether the card number length for each of the credit cards (Visa, MasterCard and American Express) is of the proper length.

FIG. 8 illustrates a client markup that generates the same GUI of FIG. 6 for obtaining credit card information to be provided to web server 204 using speech recognition. Although speech recognition will be discussed below with respect to FIGS. 8-14, it should be understood that the techniques described can be similarly applied in handwriting recognition, gesture recognition and image recognition.

Generally, the extensions (also commonly known as "tags") are a small set of XML elements, with associated attributes and DOM object properties, events and methods, which may be used in conjunction with a source markup document to apply a recognition interface, DTMF or call control to a source page. The extensions formalities and semantics are independent of the nature of the source

document, so the extensions can be used equally effectively within HTML, XHTML, XML, WML, or with any other SGML-derived markup. The extension follow the document object model wherein new functional objects or elements, which can be hierarchical, are provided. Each of the elements are discussed in detail in the Appendix, but generally the elements can include attributes, properties, methods, events and/or other "child" elements.

At this point, it should also be noted that the extensions may be interpreted in two different "modes" according to the capabilities of the device upon which the browser is being executed on. In a first mode, "object mode", the full capabilities are available. The programmatic manipulation of the extensions by an application is performed by whatever mechanisms are enabled by the browser on the device, e.g. a JavaScript interpreter in an XHTML browser, or a WMLScript interpreter in a WML browser. For this reason, only a small set of core properties and methods of the extensions need to be defined, and these manipulated by whatever programmatic mechanisms exist on the device or client side. The object mode provides eventing and scripting and can offer greater functionality to give the dialog author a much finer client-side control over speech interactions. As used herein, a browser that supports full event and scripting is called an "uplevel browser". This form of a browser will support all the attributes, properties, methods and events of the extensions. Uplevel browsers are commonly found on devices with greater processing capabilities.

The extensions can also be supported in a "declarative mode". As used herein, a browser operating in a declarative mode is called a "downlevel browser" and does

not support full eventing and scripting capabilities. Rather, this form of browser will support the declarative aspects of a given extension (i.e. the core element and attributes), but possibly not all the DOM (document object model) object properties, methods and events. This mode employs exclusively declarative syntax, and may further be used in conjunction with declarative multimedia synchronization and coordination mechanisms (synchronized markup language) such as SMIL (Synchronized Multimedia Integration Language) 2.0. Downlevel browsers will typically be found on devices with limited processing capabilities.

At this point though, a particular mode of entry should be discussed. In particular, use of speech recognition in conjunction with at least a display and, in a further embodiment, a pointing device as well to indicate the fields for data entry is particularly useful. Specifically, in this mode of data entry, the user is generally under control of when to select a field and provide corresponding information. For instance, in the example of FIG. 6, a user could first decide to enter the credit card number in field 252 and then enter the type of credit card in field 250 followed by the expiration date in field 254. Likewise, the user could return back to field 252 and correct an errant entry, if desired. When combined with speech recognition as described below, an easy and natural form of navigation is provided. As used herein, this form of entry using both a screen display allowing free form selection of fields and voice recognition is called "multi-modal".

Referring back to FIG. 8, an example of HTML mark-up language code is illustrated. Like the HTML code

illustrated in FIG. 7, this code also includes a body portion 270 and a script portion 272. Also like the code illustrated in FIG. 7, the code illustrated in FIG. 8 includes indications as to the type of action to perform as well as the location of the form. Entry of information in each of the fields 250, 252 and 254 is controlled or executed by code portions 280, 282 and 284, respectively. Referring first to code portion 280, on selection of field 250, for example, by use of stylus 33 of device 30, the event "onClick" is initiated which calls or executes function "talk" in script portion 272. This action activates a grammar used for speech recognition that is associated with the type of data generally expected in field 250. This type of interaction, which involves more than one technique of input (e.g. voice and pen-click/roller) is referred as "multimodal".

It should be noted that the speech recognition extensions exemplified in FIG. 8 are not intended to have a default visual representation on the browser of the client, since for many applications it is assumed that the author will signal the speech enablement of the various components of the page by using application-specification graphical mechanisms in the source page. Nevertheless, if visual representations are desired, the extensions can so be modified.

Referring now back to the grammar, the grammar is a syntactic grammar such as but not limited to a context-free grammar, a N-grammar or a hybrid grammar. (Of course, DTMF grammars, handwriting grammars, gesture grammars and image grammars would be used when corresponding forms of recognition are employed. As used herein, a "grammar" includes information for performing recognition, and in a

further embodiment, information corresponding to expected input to be entered, for example, in a specific field) A new control 290 (herein identified as "reco"), comprising a first extensor of the mark-up language, includes various elements, two of which are illustrated, namely a grammar element "grammar" and a "bind" element. Generally, like the code downloaded to a client from web server 202, the grammars can originate at web server 202 and be downloaded to the client and/or forwarded to a remote server for speech processing. The grammars can then be stored locally thereon in a cache. Eventually, the grammars are provided to the recognition server 204 for use in recognition. The grammar element is used to specify grammars, either inline or referenced using an attribute.

Upon receipt of recognition results from recognition server 204 corresponding to the recognized speech, handwriting, gesture, image, etc., syntax of reco control 290 is provided to receive the corresponding results and associate it with the corresponding field, which can include rendering of the text therein on display 34. In the illustrated embodiment, upon completion of speech recognition with the result sent back to the client, it deactivates the reco object and associates the recognized text with the corresponding field. Portions 282 and 284 operate similarly wherein unique reco objects and grammars are called for each of the fields 252 and 254 and upon receipt of the recognized text is associated with each of the fields 252 and 254. With respect to receipt of the card number field 252, the function "handle" checks the length of the card number with respect to the card type in a manner similar to that described above with respect to FIG. 7.

Generally, use of speech recognition in conjunction with architecture 200 and the client side mark-up language occurs as follows: first, the field that is associated with the speech to be given is indicated. In the illustrated embodiment, the stylus 33 is used; however, it should be understood that the present invention is not limited to use of the stylus 33 wherein any form of indication can be used such as buttons, a mouse pointer, rotatable wheels or the like. Corresponding event such as "onClick" can be provided as is well known with use of visual mark-up languages. It should be understood that the present invention is not limited to the use of the "onClick" event to indicate the start of voice, handwriting, gesture, etc commands. Any available GUI event can be used for the same purpose as well, such as "onSelect". In one embodiment, such eventing is particularly useful for it serves to indicate both the beginning and/or end of the corresponding speech. It should also be noted that the field for which the speech is directed at can be indicated by the user as well as programs running on the browser that keep track of user interactions.

At this point, it should be stated that different scenarios of speech recognition require different behaviors and/or outputs from recognition server 204. Although the starting of the recognition process is standard in all cases - an explicit start () call from uplevel browsers, or a declarative <rec> element in downlevel browsers - the means for stopping speech recognition may differ.

In the example above, a user in a multimodal application will control input into the device by, for example, tapping and holding on a pressure sensitive

display. The browser then uses a GUI event, e.g. "pen-up", to control when recognition should stop and then returns the corresponding results. However, in a voice-only scenario such as in a telephone application (discussed below) or in a hands-free application, the user has no direct control over the browser, and the recognition server 204 or the client 30, must take the responsibility of deciding when to stop recognition and return the results (typically once a path through the grammar has been recognized). Further, dictation and other scenarios where intermediate results need to be returned before recognition is stopped (also known as "open microphone") not only requires an explicit stop function, but also needs to return multiple recognition results to the client 30 and/or web server 202 before the recognition process is stopped.

In one embodiment, the Reco element can include a "mode" attribute to distinguish the following three modes of recognition, which instruct the recognition server 204 how and when to return results. The return of results implies providing the "onRecd" event or activating the "bind" elements as appropriate. In one embodiment, if the mode is unspecified, the default recognition mode can be "automatic".

FIG. 12 is a pictorial representation of operation of the "automatic" mode for speech recognition (similar modes, events, etc. can be provided for other forms of recognition). A timeline 281 indicates when the recognition server 204 is directed to begin recognition at 283, and where the recognition server 204 detects speech at 285 and determines that speech has ended at 287.

Various attributes of the Reco element control behavior of the recognition server 204. The attribute

"initialTimeout" 289 is the time between the start of recognition 283 and the detection of speech 285. If this time period is exceeded, "onSilence" event 291 will be provided from the recognition server 204, signaling that recognition has stopped. If the recognition server 204 finds the utterance to be unrecognizable, an "onNoReco" event 293 will be issued, which will also indicate that recognition has stopped.

Other attributes that can stop or cancel recognition include a "babbleTimeout" attribute 295, which is the period of time in which the recognition server 204 must return a result after detection of speech at 285. If exceeded, different events are issued according to whether an error has occurred or not. If the recognition server 204 is still processing audio, for example, in the case of an exceptionally long utterance, the "onNoReco" attribute 293 is issued. However, if the "babbleTimeout" attribute 295 is exceeded for any other reason, a recognizer error is more likely and an "onTimeout" event 297 is issued. Likewise, a "maxTimeout" attribute 299 can also be provided and is for the period of time between the start of recognition 283 and the results returned to the client 30. If this time period is exceeded, the "onTimeout" event 297 is issued.

If, however, a time period greater than an "endSilence" attribute 301 is exceeded, implying that recognition is complete, the recognition server 204 automatically stops recognition and returns its results. It should be noted that the recognition server 204 can implement a confidence measure to determine if the recognition results should be returned. If the confidence measure is below a threshold, the "onNoReco" attribute 293 is issued, whereas if the confidence measure is above the

threshold a "onNoReco" attribute 303 and the results of recognition are issued. FIG. 12 thereby illustrates that in "automatic mode" no explicit stop () calls are made.

FIG. 13 pictorially illustrates "single mode" operation of the recognition server 204. Attributes and events described above with respect to the "automatic mode" are applicable and are so indicated with the same reference numbers. However, in this mode of operation, a stop () call 305 is indicated on timeline 281. The stop () call 305 would correspond to an event such as "pen-up" by the user. In this mode of operation, the return of a recognition result is under the control of the explicit stop () call 305. As with all modes of operation, the "onSilence" event 291 is issued if speech is not detected within the "initialTimeout" period 289, but for this mode of operation recognition is not stopped. Similarly, a "onNoReco" event 293 generated by an unrecognizable utterance before the stop () call 305 does not stop recognition. However, if the time periods associated with the "babbleTimeout" attribute 295 or the "maxTimeout" attribute 299 are exceeded recognition will stop.

FIG. 14 pictorially illustrates "multiple mode" operation of the recognition server 204. As indicated above, this mode of operation is used for an "open-microphone" or in a dictation scenario. Generally, in this mode of operation, recognition results are returned at intervals until an explicit stop () call 305 is received or the time periods associated with the "babbleTimeout" attribute 295 or the "maxTimeout" attribute 299 are exceeded. It should be noted, however, that after any "onSilence" event 291, "onReco" event 303, or "onNoReco"

event 293, which does not stop recognition, timers for the "babbleTimeout" and "maxTimeout" periods will be reset.

Generally, in this mode of operation, for each phrase that is recognized, a "onReco" event 303 is issued and the result is returned until the stop () call 305 is received. If the "onSilence" event 291 is issued due to an unrecognizable utterance these events are reported but recognition will continue.

As indicated above, the associated reco object or objects for the field is activated, which includes providing at least an indication to the recognition server 204 of which grammar to use. This information can accompany the speech data recorded at the client 30 and sent to the recognition server 204. As indicated above, speech data can comprise streaming data associated with the speech entered by the user, or can include pre-processed speech data indicating speech features that are used during speech recognition. In a further embodiment, client side processing can also include normalization of the speech data such that the speech data received by the recognition server 204 is relatively consistent from client to client. This simplifies speech processing of the recognition server 204 thereby allowing easier scalability of the recognition server 204 since the recognition server can be made stateless with respect to the type of client and communication channel.

Upon receipt of the recognition result from the recognition server 204, the recognition result is associated with the corresponding field, and client-side verification or checking can be performed, if desired. Upon completion of all of the fields associated with the code currently rendered by the client, the information is sent

to web server 202 for application processing. From the foregoing, it should be clear that although the web server 202 has provided code or pages/scripts suitable for recognition to the client 30, the recognition services are not performed by the web server 202, but rather by the recognition server 204. The inventor, however, does not preclude an implementation where the recognition server 204 is collocated with the web server 202, or the recognition server 204 is part of the client 30. In other words, the extensions provided herein are beneficial even when the recognition server 204 is combined with the web server 202 or client 30 because the extension provides a simple and convenient interface between these components.

While not shown in the embodiment illustrated in FIG. 8, the reco control can also include a remote audio object (RAO) to direct the appropriate speech data to the recognition server 204. The benefit for making RAO a plug-in object is to allow a different one for each different device or client because the sound interface may likely be different. In addition, the remote audio object can allow multiple reco elements to be activated at the same time.

FIGS. 9A and 9B illustrate a voice-only mark-up language embodied herein as HTML with pages/scripts. As clearly illustrated, the code also includes a body portion 300 and a script portion 302. There is another extension of the markup language - prompt control 303 which include attributes like bargain. However, speech recognition is conducted differently in the voice-only embodiment of FIGS. 9A and 9B. The process is now controlled entirely by the script function "checkFilled" which will determine the unfilled fields and activate correspondent prompt and new objects. Nevertheless, grammars are activated using the

same context as that described above with respect to FIG. 8, wherein speech data and the indication of the grammar to use are provided to the recognition server 204. Likewise, the output received from the recognition server 204 is associated with fields of the client (herein telephony voice browser 212).

Other features generally unique to voice-only applications is an indication to the user when speech has not been recognized. In multimodal applications such as FIG. 8, 'onNoReco' simply puts null value on the displayed field to indicate no-recognition, thus no further action is required. In the voice-only embodiment, "onNoReco" 305 calls or executes a function "mumble", which forwards a word phrase to recognition server 204, that in turn, is converted to speech using a suitable text-to-speech system 307 (FIG. 5). Recognition server 204 returns an audio stream to the telephony voice browser 212, which in turn, is transmitted to phone 80 to be heard by the user. Likewise, other waveform prompts embodied in the voice-only application are also converted, when necessary, to an audio stream by recognition server 204.

It should be noted that in this example after playing the welcome prompt via function "welcome", function "checkFilled" prompts the user for each of the fields and activates the appropriate grammars, including repeating the fields that have been entered and confirming that the information is correct, which includes activation of a "confirmation" grammar. Note in this embodiment, each of the reco controls is initiated from the script portion 302, rather than the body portion of the previous example.

As another aspect of the present invention, the markup language executable on different types of client

devices (e.g. multimodal and non-display, voice input based client devices such as a telephone) unifies at least one of speech-related events, GUI events and telephony events for a web server interacting with each of the client devices. This is particular advantageous for it allows significant portions of the web server application to be written generically or independent of the type of client device. An example is illustrated in FIGS. 8 and 9A, 9B with the "handle" functions.

Although not shown in FIG. 9, there are two more extensions to the markup language to support telephony functionality - DTMF (Dual Tone Modulated Frequency) control and call control elements or objects. DTMF works similarly to reco control. It specifies a simple grammar mapping from keypad string to text input. For example, "1" means grocery department, "2" mean pharmacy department, etc. On the other hand, call object deals with telephony functions, like call transfer and 3rd party call. The attributes, properties, methods and events are discussed in detail in the Appendix.

FIGS. 10A and 10B illustrate yet another example of a mark-up language suitable for a voice-only mode of operation. In this embodiment, the user is allowed to have some control over when information is entered or spoken. In other words, although the system may initiate or otherwise direct the user to begin speaking, the user may offer more information than what was initially asked for. This is an example of "mixed initiative". Generally, in this form of dialog interaction, the user is permitted to share the dialog initiative with the system. Besides the example indicated above and discussed below in detail where the user provides more information then requested by a prompt,

the user could also switch tasks when not prompted to do so.

In the example of FIGS 10A and 10B, a grammar identified as "do_field" includes the information associated with the grammars "g_card_types", "g_card_num" and "g_expiry_date". In this example, telephony voice browser 212 sends speech data received from phone 80 and an indication to use the "do_field" grammar to recognition server 204 upon receipt of the recognized speech as denoted by "onReco", the function "harcie" is called or executed that includes associating the values for any or all of the fields recognized from the speech data. In other words, the result obtained from the recognition server 204 also includes indications for each of the fields. This information is parsed and associated with the corresponding fields according to binding rules specified in 405. As indicated in FIG. 5, the recognition server 204 can include a parser 309.

From FIGS. 7, 8, 9A, 9B, 10A and 10B, a very similar web development framework is used. Data presentation is also very similar in each of these cases. In addition, the separation of data presentation and flow controls allow maximum reusability between different applications (system initiative and mixed-initiative), or different modalities (GUI web-based, voice-only and multimodal). This also allows a natural extension from voice-only operation through a telephone to a multimodal operation when phones include displays and functionalities similar to device 30. Appendix A provides further details of the controls and objects discussed above.

As indicated above, uplevel browsers can use scripting in order to perform various needs such as

invoking the function "handle" in the above examples to assign the recognition results. In the embodiments described above and as further described in Appendix A at 2.1.2, the "bind" element will parse the recognition results and assign values wherein the "bind" element is a subelement or child element of the "reco" element.

Although scripting can be useful, it is believed by many not to be always the best form of browser implementation due, for example, to security concerns. Therefore, in yet another embodiment or aspect of the present invention, the "bind" element is a high level element (similar to element "reco") and is provided with other richer properties, which can in effect mimic scripting without scripting per se.

Without using scripting or without using the aspect of the present invention discussed below, some of the capabilities discussed below such as sophisticated dialog effects could only be accomplished by submitting a page back to the web server 202, executing the application logic thereon to generate a new page, and sending the page back to the client device. This aspect of the present invention allows a programmer to invoke methods on objects of the page without incurring a server roundtrip.

In the embodiments described above, the "bind" element has only the attributes "TargetElement" and "TargetAttribute" for assigning recognition results to a field in the form or web page. In a further embodiment, the "bind" element also includes a "TargetMethod", which is added for object method invocation. The use and capability of "TargetMethod" is the principal technique for mimicking scripting. For example, the following syntax can be used to invoke the "X" method of the object "OBJ1":

<bind TargetElement = "OBJ1" TargetMethod = "X"...>.

Note that although the examples shown here follow the HTML/XHTML event syntax, those skilled at art should find it straightforward to generalize the <bind> usage to other eventing mechanisms, including but not limited to W3C Document Object Model Level 2 or Level 3 eventing standard, ECMA Common language Infrastructure (CLI) event model, Java programming language event model, W3C Synchronous Multimedia Integration Language (SMIL), and the upcoming W3C XML Events standard proposal.

FIGS. 15A and 15B are a page of mark-up language executable on a client, particularly a downlevel browser. In this example, the user is asked through audio prompts for a drink. The system then confirms what drink was ordered. Depending on the recognition results, the "bind" element guides the execution using declared logic. When the drink is confirmed, the form is submitted back to the web server 202, all without scripting.

Generally, the mark-up example of FIGS. 15A and 15B includes a data section 350, a speech section 352 and user interface sections 354, 356 and 358. Section 354 receives the recognition result from the general inquiry as to what drink the user would like and directs interactive recognition flow to either reprompting, inquiring as to whether cream or sugar is needed, or confirming the drink ordered. In particular, section 356 receives a recognition result when cream or sugar is also ordered. Section 358 receives the recognition result for confirmation of the drink. Section 360 is a call control section employing a new messaging object "SMEX", which is discussed further below.

As indicated above, the "bind" element of this aspect of the present invention includes object method invocation, which will initiate user interaction in the example of FIGS. 15A and 15B by playing the "welcome" prompt when the "start" method on the "welcome" object is executed at 361.

The user is then asked "Do you want coke, coffee or orange juice?" by executing the "start" method of the "asked" object at 362. Recognition is then performed by invoking the "start" method on the recognition "reco_drink" object at 363.

The mark-up of section 354 is then executed wherein the grammar used by the recognition server 204 is provided by the Xpath statement ".//drink types". Note that although this example utilizes W3C Xpath language, those skilled at art should find it straightforward to extend the concept to other standard languages, including but not limited to W3C XML Query Language (XQL). As specified by the "bind" element 364, if the recognition result received from the recognition server 204 has a confidence score less than 10, the prompt object "reprompt" is executed at 366, followed by the prompt object "ask" 368, at which point, the recognition object "reco_drink" is reinitiated at 370. If the recognition result returned is "coffee" having a confidence greater than 10, the field "drink" is assigned the value of the recognition result at 372 and the user is then prompted as to whether or not he/she would like cream or sugar by the prompt object "cream_sugar" at 374. The recognition object "reco_cream_sugar" in section 356 is then invoked at 376. Otherwise, if the confidence score of the recognition result is greater than 10, but not coffee, the field drink is then again assigned at 378. Confirmation

of the recognition result is provided at 380 by executing the prompt object "confirm", followed by invocation of the recognition object "reco.yesno" in section 358 at 382. If the user answers "yes" with a confidence score greater than 10, the prompt object "thanks" is played at 384 and then the form is then submitted at 386. Otherwise, if the user answers "no" or the confidence score of the recognition result is less than 10, the prompt object "retry" is executed at 390, once again followed by the prompt object "ask" being executed at 392, and invocation of the "reco_drink" recognition object at 394.

From the foregoing example, the "bind" element allows multiple invocations of methods as indicated in sections 354, 356 or 358. If desired, multiple assignments of the recognized result can also be declared. In the embodiment illustrated, when multiple assignments and method invocations are declared, they are executed in the document order.

In a further embodiment, a convention for passing method arguments is also provided. In other words, some methods may require a list of arguments. This is achieved by using "arg" subelement. For example, given the following

```
mark-up:
<bind TargetElement = "OBJ" TargetMethod =
  "F"><arg>X</arg><arg>Y</arg></bind>
```

is equivalent to "OBJ.F(X,Y)", or "OBJ" is an object that has a method "F" with parameters or arguments "X" and "Y".

The "bind" element can also include an "event" attribute, which declares the bind element is intended for which event. For example, the mark-up:

```
<bind event = "onNoReco" = TargetElement =
  "prompt1" TargetMethod = "start"/>
```

means when the "onNoReco" event is sent, the method "start" of the object "prompt1" will be invoked. To be consistent with use of the "bind" element as a child element of the "Reco" element as described above for example with respect to FIG. 8, the default attribute for the "bind" element is "onReco".

The "bind" element as a high level element can include any of the events specified in section 2.4 of the Appendix. In addition, the "bind" element can also include a "onError" event having a "status" attribute that can be accessed and used to direct program flow. To the extent other events of the "bind" element have "status" attributes, these can be accessed as well.

In addition to checking the conditions of the recognition result, the current document or page being executed can also be checked as well. In particular, both the "test" and the "value" attribute can be extended to include a "host" primitive that refers to the root node of the containing document. For example, referring back to FIGS. 15A, 15D, the example contained therein has additional logic at section 354 to ask whether the user wants cream or sugar when he/she asks for coffee. The flags for adding cream or sugar and thus invocation of section 356, will be turned on only if the drink field is "coffee" as specified by the markup

```
"host()/get_drink/drink='coffee'".
```

It should also be noted, the "bind" element is not only applicable to recognition results from the speech server 204 and receive or assign values in the document, but also to message objects (herein denoted by "smex", for example, from applications running on the client device. In the example of FIGS. 15A and 15B, the page is executed when

a telephony application running on the client device detects a call. In section 360, the "bind" element executes or plays the "welcome" prompt and begins recognition by executing the "reco_drink" object, when the message "/Call_connected" is received. Like the recognition results received from the speech server 204, the messages received can vary greatly. Some of the messages are well-defined in order to initiate desired program flow. Others can be received and processed (e.g. parsed just like received recognition results of the recognition server. For example, this allows the markup to be used like a natural language parser of text from a keyboard. The reco element in Appendix A includes a property for performing this function. Likewise, the prompt element can be used to provide a textual message for dynamic content or audio wave files by using the property "innertext", also further explained in the Appendix A. Eventing can be similar to eventing for recognition results. For example, eventing can include "onReceived", which gets sent when the message source (e.g. application running on the client device) has a message available for the browser.

The "smex" or message object thus allows the markup tags discussed herein to be extended to other components or applications running on the client device. As another example, the message object can be used to communicate to a TTY component for the hearing impaired running on the client device. Rather than using speech recognition, the TTY component will provide a message of what the user has typed. This message is then used as if a recognition result was received from the recognition server wherein the message can be parsed and assigned to fields of the form, or other processing can take place using the

"reco", "grammar" or "bind" elements described above. Further discussion of the message or "smex" object is provided in Appendix A.

The "bind" element can also include a "for" attribute, which allows its action to be attached to other objects on the page. For example, a mark-up such as:

```
<bind for = "prompt 1" event = "onComplete"
targetElement = "prompt 2" = targetMethod =
"start" />
```

will invoke the start method of the object "prompt 2" when the object "prompt 1" sends the event "onComplete".

Referring back to FIG. 5, web server 202 can include a server side plug-in declarative authoring tool or module 320 (e.g. ASP or ASP+ by Microsoft Corporation, JSP, or the like). Server side plug-in module 320 can dynamically generate client-side mark-ups and even a specific form of mark-up for the type of client accessing the web server 202. The client information can be provided to the web server 202 upon initial establishment of the client/server relationship, or the web server 202 can include modules or routines to detect the capabilities of the client. In this manner, server side plug-in module 320 can generate a client side mark-up for each of the voice recognition scenarios, i.e. voice only through phone 80 or multimodal for device 30. By using a consistent client side model (reco and prompt controls that can be used in each application), application authoring for many different clients is significantly easier.

In addition to dynamically generating client side mark-ups, high-level dialog modules, like getting credit card information illustrated in FIG. 6 with a mark-up examples of FIGS. 8, 9A and 9B, can be implemented as a

server-side control as stored in store 324 for use by developers in application authoring. In general, the high-level dialog modules 324 would generate dynamically client-side markup and script in both voice-only and multimodal scenarios based on parameters specified by developers. The high-level dialog modules can include parameters to generate client-side mark-ups to fit the developers' needs. For example, a credit card information module can include a parameter indicating what types of credit cards the client-side mark-up script should allow. A sample ASP+ page using in server side plug-in module 320 is illustrated in FIG. 11.

Although the present invention has been described with reference to preferred embodiments, workers skilled in the art will recognize that changes may be made in form and detail without departing from the spirit and scope of the invention.

APPENDIX A

1 Introduction

The following tags are a set of markup elements that allows a document to use speech as an input or output medium. The tags are designed to be self-contained XML that can be imbedded into any SGML derived markup languages such as HTML, XHTML, CHTML, SMIL, WML and the like. The tags used herein are similar to GAPI 5.0, which are known methods available from Microsoft Corporation of Redmond, Washington. The tags, elements, events, attributes, properties, return values, etc. are merely exemplary and should not be considered limiting. Although exemplified herein for speech and DTMF recognition, similar tags can be provided for other forms of recognition.

The main elements herein discussed are:

```

<prompt ...>    for speech synthesis configuration and
prompt playing
<reco ...>      for recognizer configuration and recognition
execution and post-processing
<grammar ...>   for specifying input grammar resources
<bind ...>      for processing of recognition results
<dtmf ...>      for configuration and control of DTMF

```

2 Reco

The Reco element is used to specify possible user inputs and a means for dealing with the input results. As such, its main elements can be <grammar> and <bind>, and it contains resources for configuring recognizer properties.

Reco elements are activated programmatically in uplevel browsers via Start and Stop methods, or in SMIL-enabled browsers by using SMIL commands. They are considered active declaratively in downlevel browsers (i.e. non script-supporting browsers) by their presence on the page. In order to permit the activation of multiple grammars in parallel, multiple Reco elements may be considered active simultaneously.

Recos may also take a particular mode - 'automatic', 'single' or 'multiple' - to distinguish the kind of recognition scenarios which they enable and the behavior of the recognition platform.

2.1 Reco content

The Reco element contains one or more grammars and optionally a set of bind elements which inspect the results of recognition and copy the relevant portions to values in the containing page.

In uplevel browsers, Reco supports the programmatic

activation and deactivation of individual grammar rules.

Note also that all top-level rules in a grammar are active by default for a recognition context.

2.1.1 <grammar> element

The grammar element is used to specify grammars, either inline or referenced using the src attribute. At least one grammar (either inline or referenced) is typically specified. Inline grammars can be text-based grammar formats, while referenced grammars can be text-based or binary type. Multiple grammar elements may be specified. If

more than one grammar element is specified, the rules within grammars are added as extra rules within the same grammar. Any rules with the same name will be over-written.

Attributes:

- **src**: Optional if inline grammar is specified. URI of the grammar to be included. Note that all top-level rules in a grammar are active by default for a recognition context.
- **langID**: Optional. String indicating which language speech engine should use. The string format follows the xml:lang definition. For example, langID="en-us" denotes US English. This attribute is only effective when the langID is not specified in the grammar URI. If unspecified, defaults to US English.

If the langID is specified in multiple places then langID follows a precedence order from the lowest scope - remote grammar file (i.e language id is specified within the grammar file) followed by grammar element followed by reco element.

```
<grammar src="FromCity.xml" />
or
<grammar>
  <rule topLevel="active">
    <p>from </p>
    <rule ref name="cities" />
  </rule>
  <rule name="cities">
    <l>
      <p> Cambridge </p>
      <p> Seattle </p>
      <p> London </p>
    </l>
  </rule>
</grammar>
```

</grammar>

If both a src-referenced grammar and an inline grammar are specified, the inline rules are added to the referenced rules, and any rules with the same name will be overwritten.

2.1.2 <bind> element

The bind element is used to bind values from the recognition results into the page.

The recognition results consumed by the bind element can be an XML document containing a semantic markup language (SML) for specifying recognition results. Its contents include semantic values, actual words spoken, and confidence scores. SML could also include alternate recognition choices (as in an N-best recognition result). A sample SML document for the utterance "I'd like to travel from Seattle to Boston" is illustrated below:

```
<sml confidence="40">
  <travel text="I'd like to travel from Seattle to
    Boston">
    <origin_city confidence="45"> Seattle
  </origin_city>
  <dest_city>
    <dest_city confidence="35"> Boston
  </dest_city>
  </travel>
</sml>
```

Since an in-grammar recognition is assumed to produce an XML document - in semantic markup language, or SML - the values to be bound from the SML document are referenced using an XPath query. And since the elements in the page into which the values will be bound should be are uniquely identified

(they are likely to be form controls), these target elements are referenced directly.

Attributes:

- **targetElement:** Required. The element to which the value content from the SML will be assigned (as in W3C SML 2.0).
- **targetAttribute:** Optional. The attribute of the target element to which the value content from the SML will be assigned (as with the attributeName attribute in SML 2.0). If unspecified, defaults to "value".
- **test:** Optional. An XML Pattern (as in the W3C XML DOM specification) string indicating the condition under which the recognition result will be assigned. Default condition is true.
- **value:** Required. An XPATH (as in the W3C XML DOM specification) string that specifies the value from the recognition result document to be assigned to the target element.

Example:

So given the above SML return, the following reco element uses bind to transfer the values in origin_city and dest_city into the target page elements textBoxOrigin and textBoxDest:

```
<input name="textBoxOrigin" type="text"/>
<input name="textBoxDest" type="text" />

<reco id="travel">
  <grammar src="/city.xml" />
```

```

<bind targetElement="textBoxOrigin"
value="//origin_city" />
<bind targetElement="textBoxDest"
value="//dest_city" />
</reco>

```

This binding may be conditional, as in the following example, where a test is made on the confidence attribute of the dest_city result as a pre-condition to the bind operation:

```

<bind targetElement="textBoxDest"
value="//dest_city"
test="//sml/dest_city{@confidence > 40}"
/>

```

The bind element is a simple declarative means of processing recognition results on downlevel or uplevel browsers. For more complex processing, the reco DOM object supported by uplevel browsers implements the onReco event handler to permit programmatic script analysis and post-processing of the recognition return.

2.2 Attributes and properties

The following attributes are supported by all browsers, and the properties by uplevel browsers.

2.2.1 Attributes

The following attributes of Reco are used to configure the speech recognizer for a dialog turn.

- **initialTimeout:** Optional. The time in milliseconds between start of recognition and the detection of speech. This value is passed to the recognition platform, and if exceeded, an onSilence event will be provided from the recognition platform (see 2.4.2). If not specified, the speech platform will use a default value.
- **babbleTimeout:** Optional. The period of time in milliseconds in which the recognizer must return a result after detection of speech. For recos in automatic and single mode, this applies to the period between speech detection and the stop call. For recos in 'multiple' mode, this timeout applies to the period between speech detection and each recognition return - i.e. the period is restarted after each return of results or other event. If exceeded, different events are thrown according to whether an error has occurred or not. If the recognizer is still processing audio - eg in the case of an exceptionally long utterance - the onNoReco event is thrown, with status code 13 (see 2.4.4). If the timeout is exceeded for any other reason, however, a recognizer error is more likely, and the onTimeout event is thrown. If not specified, the speech platform will default to an internal value.
- **maxTimeout:** Optional. The period of time in milliseconds between recognition start and results returned to the browser. If exceeded, the onTimeout event is thrown by the browser - this caters for network or recognizer failure in distributed

environments. For recos in 'multiple' mode, as with babbleTimeout, the period is restarted after the return of each recognition or other event. Note that the maxTimeout attribute should be greater than or equal to the sum of initialTimeout and babbleTimeout. If not specified, the value will be a browser default.

- **endsilence:** Optional. For Recos in automatic mode, the period of silence in milliseconds after the end of an utterance which must be free of speech after which the recognition results are returned. Ignored for recos of modes other than automatic. If unspecified, defaults to platform internal value.
- **reject:** Optional. The recognition rejection threshold, below which the platform will throw the 'no reco' event. If not specified, the speech platform will use a default value. Confidence scores range between 0 and 100 (integer). Reject values lie in between.
- **server:** Optional. URI of speech platform (for use when the tag interpreter and recognition platform are not co-located). An example value might be `server-protocol://yourspeechplatform`. An application writer is also able to provide speech platform specific settings by adding a querystring to the URI string, eg `protocol://yourspeechplatform?bargainEnergyThreshold=0.5`.
- **langID:** Optional. String indicating which language speech engine should use. The string format follows the xml:lang definition. For example, langID="en-us"

denotes US English. This attribute is only effective when the langID is not specified in the grammar element (see 2.1.1).

- **mode:** Optional. String specifying the recognition mode to be followed. If unspecified, defaults to "automatic" mode.

2.2.2 Properties

The following properties contain the results returned by the recognition process (these are supported by uplevel browsers).

- **resultCode** Read-only. The results of recognition, held in an XML DOM node object containing semantic markup language (SML), as described in 2.1.2. In case of no recognition, the property returns null.
- **text** Read/Write. A string holding the text of the words recognized (i.e., a shorthand for contents of the text attribute of the highest level element in the SML recognition return in resultCode in a read mode. In write mode, a string can be assigned, which will then be parsed as if the string corresponds to the recognition result. The write mode allows extension of the markup tags and processing thereof to other components or applications on the client device. The string can be obtained from the "smex" message object.
- **status:** Read-only. Status code returned by the recognition platform. Possible values are 0 for successful recognition, or the failure values -1 to -4 (as defined in the exceptions possible on the Start method (section 2.3.1) and Activate method (section

2.3.4)), and statuses -11 to -15 set on the reception of recognizer events (see 2.4).

2.3 Object methods

Reco activation and grammar activation may be controlled using the following methods in the Reco's DOM object. With these methods, uplevel browsers can start and stop Reco objects, cancel recognitions in progress, and activate and deactivate individual grammar top-level rules (uplevel browsers only).

2.3.1 Start

The Start method starts the recognition process, using as active grammars all the top-level rules for the recognition context which have not been explicitly deactivated.

Syntax:

```
Object.Start( )
```

Return value:

None.

Exception:

The method sets a non-zero status code and fires an onNoReco event when fails. Possible failures include no grammar (reco status = -1), failure to load a grammar, which could be a variety of reasons like failure to compile grammar, non-existent URI (reco status = -2), or speech platform errors (reco status = -3).

2.3.2 Stop

The Stop method is a call to end the recognition process. The Reco object stops recording audio, and the recognizer

returns recognition results on the audio received up to the point where recording was stopped. All the recognition resources used by Reco are released, and its grammars deactivated. (Note that this method need not be used explicitly for typical recognitions in automatic mode, since the recognizer itself will stop the reco object on endpoint detection after recognizing a complete sentence.) If the Reco has not been started, the call has no effect.

Syntax:

```
Object.Stop( )
```

Return value:

None.

Exception:

None.

2.3.3 Cancel

The Cancel method stops the audio feed to the recognizer, deactivates the grammar and releases the recognizer and discards any recognition results. The browser will disregard a recognition result for canceled recognition. If the recognizer has not been started, the call has no effect.

Syntax:

```
Object.Cancel( )
```

Return value:

None.

Exception:

None.

2.3.4 Activate

The Activate method activates a top-level rule in the context free grammar (CFG). Activation must be called before recognition begins, since it will have no effect during a 'Started' recognition process. Note that all the grammar top-level rules for the recognition context which have not been explicitly deactivated are already treated as active.

Syntax:
Object.Activate(strName);
Parameters:
o strName: Required. Rule name to be activated.
Return value:
None.
Exception:
None.

2.3.5 Deactivate

The method deactivates a top-level rule in the grammar. If the rule does not exist, the method has no effect.

Syntax:
Object.Deactivate(strName);
Parameters:
o strName: Required. Rule name to be deactivated. An empty string deactivates all rules.
Return value:
None.
Exception:
None.

2.4 Reco events

The Reco DOM object supports the following events, whose handlers may be specified as attributes of the reco element.

2.4.1 onReco:

This event gets fired when the recognizer has a recognition result available for the browser. For recos in automatic mode, this event stops the recognition process automatically and clears resources (see 2.3.2). OnReco is typically used for programmatic analysis of the recognition result and processing of the result into the page.

Syntax:

Inline HTML	<Reco onReco = "handler" >
Event property	Object.onReco = handler;
	Object.onReco =
	GetRef("handler");

Event Object Info:

Bubbles	No
To invoke	User says something
Default action	Return recognition result object

Event Properties:
Although the event handler does not receive properties directly, the handler can query the event object for data (see the use of the event object in the example below).

Example

The following XHTML fragment uses onReco to call a script to parse the recognition outcome and assign the values to the proper fields.

```
<input name="textBoxOrigin" type="text" />
<input name="textBoxDest" type="text" />
<reco onReco="processCityRecognition()" />
<grammar src="/grammars/cities.xml" />
</reco>

<script><![CDATA:
    function processCityRecognition () {
        smlResult = event.srcElement.recoResult;

        origNode =
            smlResult.selectSingleNode("//origin_city");
        if (origNode != null) textBoxOrigin.value =
            origNode.text;

        destNode =
            smlResult.selectSingleNode("//dest_city");
        if (destNode != null) textBoxDest.value =
            destNode.text;
    }
</script>
```

2.4.2 onSilence:
onSilence handles the event of no speech detected by the recognition platform before the duration of time specified in the initialTimeout attribute on the Reco (see 2.2.1).

This event cancels the recognition process automatically for the automatic recognition mode.

Syntax:

Inline HTML	<reco onSilence="handler" ...>
Event property (in ECMAScript)	Object.onSilence = handler Object.onSilence = GetRef("handler");

Event Object Info:

Bubbles	No
To invoke	Recognizer did not detect speech within the period specified in the initialTimeout attribute.
Default action	Set status = -1.

Event Properties:

Although the event handler does not receive properties directly, the handler can query the event object for data.

2.4.3 onTimeout

onTimeout handles two types of event which typically reflect errors from the speech platform.

- It handles the event thrown by the tags interpreter which signals that the period specified in the maxtime attribute (see 2.2.1) expired before recognition was completed. This event will typically reflect problems that could occur in a distributed architecture.

- It also handles (ii) the event thrown by the speech recognition platform when recognition has begun but processing has stopped without a recognition within the period specified by babbleTimeout (see 2.2.1).

This event cancels the recognition process automatically.

Syntax:

Inline HTML:	<reco onTimeout="handler" ->
Event property (in ECMAScript):	Object.onTimeout = handler Object.onTimeout = GetRef("handler");

Event Object Info:

Bubbles	No
To invoke	Thrown by the browser when the period set by the maxtime attribute expires before recognition is stopped.
Default action	Set reco status to -12.

Event Properties:

Although the event handler does not receive properties directly, the handler can query the event object for data.

2.4.4 onNoReco:

onNoReco is a handler for the event thrown by the speech recognition platform when it is unable to return valid recognition results. The different cases in which this may

happen are distinguished by status code. The event stops the recognition process automatically.

Syntax:

Inline HTML:	<reco onNoReco="handler" >
Event property	Object.onNoReco = handler; Object.onNoReco = GetRef("handler");

Event Object Info:

Bubbles	No
To invoke	Recognizer detects sound but is unable to interpret the utterance.
Default action	Set status property and return null recognition result. Status codes are set as follows: status -13: sound was detected but no speech was able to be interpreted; status -14: some speech was detected and interpreted but rejected with insufficient confidence (for threshold setting, see the reject attribute in 2.2.1). status -15: speech was detected and interpreted, but a complete recognition was unable to be returned between the detection of speech and the duration specified in the babbleTimeout attribute (see 2.2.1).

Event Properties:

Although the event handler does not receive properties directly, the handler can query the event object for data.

3 Prompt

The prompt element is used to specify system output. Its content may be one or more of the following:

- inline or referenced text, which may be marked up with prosodic or other speech output information;
- variable values retrieved at render time from the containing document;
- links to audio files.

Prompt elements may be interpreted declaratively by downlevel browsers (or activated by SMIL commands), or by object methods on uplevel browsers.

3.1 Prompt content

The prompt element contains the resources for system output, either as text or references to audio files, or both.

Simple prompts need specify only the text required for output, eg:

```
<prompt id="Welcome">
  Thank you for calling ACME weather report.
</prompt>
```

This simple text may also contain further markup of any of the kinds described below.

3.1.1 Speech Synthesis markup

Any format of speech synthesis markup language can be used inside the prompt element. (This format may be specified in the 'tts' attribute described in 3.2.1.) The following example shows text with an instruction to emphasize certain words within it:

```
<prompt id="giveBalance">
  You have <emph>five dollars </emph> left in your
  account.
</prompt>
```

3.1.2 Dynamic content

The actual content of the prompt may need to be computed on the client just before the prompt is output. In order to confirm a particular value, for example, the value needs to be dereferenced in a variable. The value element may be used for this purpose.

Value Element
value: Optional. Retrieves the values of an element in the document.

Attributes:

- **targetElement:** Optional. Either href or targetElement must be specified. The id of the element containing the value to be retrieved.
- **targetAttribute:** Optional. The attribute of the element from which the value will be retrieved.

- href: Optional. The URI of an audio segment. href will override targetElement if both are present.

The targetElement attribute is used to reference an element within the containing document. The content of the element whose id is specified by targetElement is inserted into the text to be synthesized. If the desired content is held in an attribute of the element, the targetAttribute attribute may be used to specify the necessary attribute on the targetElement. This is useful for dereferencing the values in HTML form controls, for example. In the following illustration, the "value" attributes of the "textBoxOrigin" and "textBoxDest" elements are inserted into the text before the prompt is output

```
<prompt id="Confirm">
  Do you want to travel from
  <value targetElement="textBoxOrigin"
targetAttribute="value" />
  to
  <value targetElement="textBoxDest"
targetAttribute="value" />
  ?
</prompt>
```

3.1.3 Audio files

The value element may also be used to refer to a pre-recorded audio file for playing instead of, or within, a synthesized prompt. The following example plays a beep at the end of the prompt:

```
<prompt>
```

After the beep, please record your message.

```
<value href="/wav/beep.wav" />
</prompt>
```

3.1.4 Referenced prompts

Instead of specifying content inline, the src attribute may be used with an empty element to reference external content via URI, as in:

```
<prompt id="Welcome" src="/ACMEWeatherPrompts#Welcome"
/>
```

The target of the src attribute can hold any or all of the above content specified for inline prompts.

3.2 Attributes and properties

The prompt element holds the following attributes (downlevel browsers) and properties (downlevel and uplevel browsers).

3.2.1 Attributes

- tts: Optional. The markup language type for text-to-speech synthesis. Default is "SAPI 5".
- src: Optional if an inline prompt is specified. The URI of a referenced prompt (see 3.1.4).
- bargain: Optional. Integer. The period of time in milliseconds from start of prompt to when playback can be interrupted by the human listener. Default is infinite, i.e., no bargain is allowed. Bargain=0 allows immediate bargain. This applies to whichever kind of bargain is supported by platform. Either keyword or energy-based bargain times can be

configured in this way, depending on which is enabled at the time the reco is started.

- **prefetch:** Optional. A Boolean flag indicating whether the prompt should be immediately synthesized and cached at browser when the page is loaded. Default is false.

3.2.2 Properties

Uplevel browsers support the following properties in the prompt's DOM object.

- **bookmark:** Read-only. A string object recording the text of the last synthesis bookmark encountered.
- **status:** Read-only. Status code returned by the speech platform.
- **Innertext:** Read-only. This property will provide the text transcription of the prompt, which would be sent to the synthesizer. For instance, if a prompt comprises playing an audio wave file, this property provides a text version of that prompt (often stored with the audio wave file), which can then be displayed, or otherwise used, for example, by providing the text version of the prompt to a component or application running on the client device. The innertext property can also be used to provide text versions of prompts containing dynamic content.

3.3 Prompt methods

Prompt playing may be controlled using the following methods in the prompt's DOM object. In this way, uplevel browsers can start and stop prompt objects, pause and resume prompts

in progress, and change the speed and volume of the synthesized speech.

3.3.1 Start

Start playback of the prompt. Unless an argument is given, the method plays the contents of the object. Only a single prompt object is considered 'started' at a given time, so if Start is called in succession, all playbacks are played in sequence.

Syntax:

```
Object.Start([strText]);
```

Parameters:

- **strText:** the text to be sent to the synthesizer. If present, this argument overrides the contents of the object.

Return value:

None.

Exception:

Set status = -1 and fires an onComplete event if the audio buffer is already released by the server.

3.3.2 Pause

Pause playback without flushing the audio buffer. This method has no effect if playback is paused or stopped.

Syntax:

```
Object.Pause();
```

Return value:

None.

Exception:

None.

3.3.3 Resume

Resume playback without flushing the audio buffer. This method has no effect if playback has not been paused.

Syntax:

```
Object.Resume( );
```

Return value:

None.

Exception:

Throws an exception when resume fails.

3.3.4 Stop

Stop playback, if not already, and flush the audio buffer. If the playback has already been stopped, the method simply flushes the audio buffer.

Syntax:

```
Object.Stop( );
```

Return value:

None.

Exception:

None.

3.3.5 Change

Change speed and/or volume of playback. Change may be called during playback.

Syntax:

```
Object.Change(speed, volume);
```

Parameters:

o speed: Required. The factor to change. Speed=2.0 means double the current rate, speed=0.5 means

halve the current rate, speed=0 means to restore the default value.

o volume: Required. The factor to change. Volume=2.0 means double the current volume, volume=0.5 means halve the current volume, volume=0 means to restore the default value.

Return value:

None.

Exception:

None.

3.3.6 Prompt control example

The following example shows how control of the prompt using the methods above might be authored for a platform which does not support a keyword barge-in mechanism.

```
<html>
<title>Prompt control</title>
<head>
<script>
<!--
function checkKBargain() {
    news.change(1.0, 0.5); // turn down the volume
    while verifying
    if (keyword.text == "") { // result is below
        threshold
        news.change(1.0, 2.0); // restore the volume
        keyword.start(); // restart the recognition
    } else {
        news.stop(); // keyword detected! Stop the
        prompt // Do whatever that is necessary
    }
}
// </script>
<script for="window" event="onload">
<!--
```



```
news.Start(); keyword.Start();  
//  
</script>  
</head>  
<body>  
...<prompt id="news" bargain="0">  
  Stocks turned in another lackluster performance Wednesday  
  as investors received little incentive to make any big moves  
  ahead of next week's Federal Reserve meeting. The tech-heavy  
  Nasdaq Composite Index dropped 42.51 points to close at  
  2156.76. The Dow Jones Industrial Average fell 17.05 points  
  to 10866.46 after an early-afternoon rally failed.  
...</prompt>  
</body>  
</html>
```

3.4 Prompt events

The prompt DOM object supports the following events, whose handlers may be specified as attributes of the prompt element.

3.4.1 onBookmark

Fires when a synthesis bookmark is encountered. The event does not pause the playback.

Syntax:

Inline HTML	<prompt onBookmark="handler"> ...>
Event property	Object.onBookmark = handler Object.onBookmark =

	GetRef("handler");
--	--------------------

Event Object Info:

Bubbles	No
To invoke	A bookmark in the rendered string is encountered
Default action	Returns the bookmark string

Event Properties:

Although the event handler does not receive properties directly, the handler can query the event object for data.

3.4.2 onBargain

Fires when a user's bargain-in event is detected. (Note that determining what constitutes a bargain-in event, eg energy detection or keyword recognition, is up to the platform.) A specification of this event handler does not automatically turn the bargain-in on.

Syntax:

Inline HTML	<prompt onBargain="handler"> ...>
Event property	Object.onBargain = handler Object.onBargain = SetRef("handler");

Event Object Info:

Bubbles	No
To invoke	A bargain event is encountered

Default action	None
-------------------	------

Event Properties:

Although the event handler does not receive properties directly, the handler can query the event object for data.

3.4.3 onComplete:

Fires when the prompt playback reaches the end or exceptions (as defined above) are encountered.

Syntax:

Inline HTML	<prompt onComplete="handler" ...>
Event property	Object. onComplete = handler Object. onComplete = GetRef("handler");

Event Object Info:

Bubbles	No
To invoke	A prompt playback completes
Default action	Set status = 0 if playback completes normally, otherwise set status as specified above.

Event Properties:

Although the event handler does not receive properties directly, the handler can query the event object for data.

3.4.4 Using bookmarks and events

The following example shows how bookmark events can be used to determine the semantics of a user response - either a correction to a departure city or the provision of a destination city - in terms of when bargein happened during the prompt output. The onBargein handler calls a script which sets a global 'mark' variable to the last bookmark encountered in the prompt, and the value of this 'mark' is used in the reco's postprocessing function ('heard') to set the correct value.

```
<script><![CDATA[
var mark;
function interrupt ( ) {
    mark = event.srcElement.bookmark;
}
function processCityConfirm() {
    confirm.stop(); // flush the audio buffer
    if (mark == "mark_origin_city")
        txtBoxOrigin.value =
            event.srcElement.text;
    else
        txtBoxDest.value =
            event.srcElement.text;
}
]]></script>
<body>
<input name="txtBoxOrigin" value="Seattle"
type="text"/>
<input name="txtBoxDest" type="text" />
...
<prompt id="confirm" onBargein="interrupt()"
bargein="0">
    From <bookmark mark="mark_origin_city" />
    <value targetElement="origin"
targetAttributes="value" />,
    please say <bookmark mark="mark_dest_city" /> the
    destination city you want to travel to.
</prompt>
<reco onReco="processCityConfirm()" >
    <grammar src="/grm/1033/cities.xml" />
</reco>
```

```

    </body>

```

4 DTMF

Creates a DTMF recognition object. The object can be instantiated using inline markup language syntax or in scripting. When activated, DTMF can cause prompt object to fire a bargin-in event. It should be noted the tags and eventing discussed below with respect to DTMF recognition and call control discussed in Section 5 generally pertain to interaction between the voice browser 216 and media server 214.

4.1 Content

- dtmfgrammar: for inline grammar.
- bind: assign DTMF conversion result to proper field.

Attributes:

- targetElement: Required. The element to which a partial recognition result will be assigned to (cf. same as in W3C SMIL 2.0).
- targetAttribute: the attribute of the target element to which the recognition result will be assigned to (cf. same as in SMIL 2.0). Default is "value".
- test: condition for the assignment. Default is true.

Example 1: map keys to text

```

<input type="text" name="city"/>

```

```

<DTMF id="city_choice" timeout="2000" numDigits="1">
  <dtmfgrammar>
    <key value="1">Seattle</key>
    <key value="2">Boston</key>
  </dtmfgrammar>
  <bind targetElement="city" targetAttribute="value" />
</DTMF>

```

When "city_choice" is activated, "Seattle" will be assigned to the input field if the user presses 1, "Boston" if 2, nothing otherwise.

Example 2: How DTMF can be used with multiple fields.

```

<input type="text" name="area_code" />
<input type="text" name="phone_number" />
<DTMF id="area_code" numDigits="3"
  onReco="extension.Activate()">
  <bind targetElement="area_code" />
</DTMF>
<DTMF id="extension" numDigits="7">
  <bind targetElement="phone_number" />
</DTMF>

```

This example demonstrates how to allow users entering into multiple fields.

Example 3: How to allow both speech and DTMF inputs and disable speech when user starts DTMF.

```

<input type="text" name="credit_card_number" />
<prompt onBookmark="ctmf.Start(); speech.Start()"
  bargin="C">
  Please say <bookmark name="starting" />
  or enter your credit card number now
</prompt>
<DTMF id="dtmf" escape="#" length="16"
  interdigitTimeout="2000"
  onKeyPress="speech.Stop()">
  <bind targetElement="credit_card_number" />

```

```

</DTMF>
<reco id="speech" >
  <grammar src="/igm/1033/digits.xml" />
  <bind targetElement="credit_card_number" />
</reco>

```

4.2 Attributes and properties

4.2.1 Attributes

- dtmfgrammar: Required. The URI of a DTMF grammar.

4.2.2 Properties

- DTMFgrammar Read-Write.

An XML DOM Node object representing DTMF to string conversion matrix (also called DTMF grammar). The default grammar is

```

<dtmfgrammar>
  <key value="0">0</key>
  <key value="1">1</key>
  ...
  <key value="9">9</key>
  <key value="*">*</key>
  <key value="#">#</key>
</dtmfgrammar>

```

- flush

Read-write, a Boolean flag indicating whether to automatically flush the DTMF buffer on the underlying telephony interface card before activation. Default is false to enable type-ahead.

- escape

Read-Write. The escape key to end the DTMF reading session. Escape key is one key.

- numDigits

Read-Write. Number of key strokes to end the DTMF reading session. If both escape and length are specified, the DTMF session is ended when either condition is met.

- dtmfResult

Read-only string, storing the DTMF keys user has entered. Escape is included in result if typed.

- text

Read-only string storing white space separated token string, where each token is converted according to DTMF grammar.

- initialTimeout

Read-Write. Timeout period for receiving the first DTMF keystroke, in milliseconds. If unspecified, defaults to the telephony platform's internal setting.

- interdigitTimeout

Read-Write. Timeout period for adjacent DTMF keystrokes, in milliseconds. If unspecified, defaults to the telephony platform's internal setting.

4.3 Object methods:

4.3.1 Start

Enable DTMF interruption and start a DTMF reading session.

4.4 Events

4.4.1 onkeypress

Fires when a DTMF key is press. This overrides the default event inherited from the HTML contrbl. When user hits the escape key, the onRec event fires, not onkeypress.

Syntax:

Inline HTML	<DTMF onkeypress="handler" ...>
Event Property	Object.onkeypress = handler
	Object.onkeypress =
	GetRef ("handler");

Event Object Info:

Bubbles	No
To invoke	Press on the touch-tone telephone key pad
Default action	Returns the key being pressed

Event Properties:

Although the event handler does not receive properties directly, the handler can query the event object for data.

4.4.2 onRec

Fires when a DTMF session is ended. The event disables the current DTMF object automatically.

Syntax:

Object.Start();

Return value:

None

Exception:

None

4.3.2 Stop

Disable DTMF. The key strokes entered by the user, however, remain in the buffer.

Syntax:

Object.Stop();

Return value:

None

Exception:

None

4.3.3 Flush

Flush the DTMF buffer. Flush can not be called during a DTMF session..

Syntax:

Object.Flush();

Return value:

None

Exception:

None

Syntax:

Inline HTML	<DTMF onReco="handler" ...>
Event property	Object.onReco = handler Object.onReco = GetRef("handler");

Event Object Info:

Bubbles	No
To invoke	User presses the escape key or the number of key strokes meets specified value.
Default action	Returns the key being pressed

Event Properties:

Although the event handler does not receive properties directly, the handler can query the event object for data.

4.4.3 onTimeout

Fires when no phrase finish event is received before time out. The event halts the recognition process automatically.

Syntax:

Inline HTML	<DTMF onTimeout="handler" ...>
Event property (in ECMAScript)	Object.onTimeout = handler Object.onTimeout = GetRef("handler");

Event Object Info:

Bubbles	No
To invoke	No DTMF key stroke is detected within the timeout specified.
Default action	None

Event Properties:

Although the event handler does not receive properties directly, the handler can query the event object for data.

5 CallControl Object

Represents the telephone interface (call, terminal, and connection) of the telephone voice browser. This object is as native as window object in a GUI browser. As such, the lifetime of the telephone object is the same as the browser instance itself. A voice browser for telephony instantiates the telephone object, one for each call. Users don't instantiate or dispose the object.

At this point, only features related to first-party call controls are exposed through this object.

5.1 Properties

address

Read-only. XML DOM node object. Implementation specific. This is the address of the caller. For PSTN, may a combination of ANI and ALI. For VoIP, this is the caller's IP address.

・ xingsBeforeAnswer

Number of rings before answering an incoming call.
Default is infinite, meaning the developer must specifically use the Answer() method below to answer the phone call. When the call center uses ACD to queue up the incoming phone calls, this number can be set to 0.

5.2 Methods

Note: all the methods here are synchronous.

5.2.1 Transfer

Transfers the call. For a blind transfer, the system may terminate the original call and free system resources once the transfer completes.

Syntax:

telephone.Transfer(strText);

Parameters:

- o strText: Required. The address of the intended receiver.

Return value:

None.

Exception:

Throws an exception when the call transfer fails. e.g., when end party is busy, no such number, fax or answering machine answers.

5.2.2 Bridge

Third party transfer. After the call is transferred, the browser may release resources allocated for the

call. It is up to the application to recover the session state when the transferred call returns using strUID. The underlying telephony platform may route the returning call to a different browser. The call can return only when the recipient terminates the call.

Syntax:

telephone.Bridge(strText, strUID, [imaxTime]);

Parameters:

- o strText: Required. The address of the intended receiver.
- o strUID: Required. The session ID uniquely identifying the current call. When the transferred call is routed back, the strUID will appear in the address attribute.
- o imaxTime: Optional. Maximum duration in seconds of the transferred call. If unspecified, defaults to platform-internal value

Return value:

None.

Exception:

None.

5.2.3 Answer

Answers the phone call.

Syntax:

telephone.Answer();

Return value:

None.

Exception:
 Throws an exception when there is no connection.
 No onAnswer event will be fired in this case.

5.2.4 Hangup

Terminates the phone call. Has no effect if no call currently in progress.

Syntax:
 telephone.Hangup();
Return value:
 None.
Exception:
 None.

5.2.5 Connect

Starts a first-party outbound phone call.

Syntax:
 telephone.Connect(strText, {iTimeout});
Parameters:
 o strText: Required. The address of the intended receiver.
 o iTimeout: Optional. The time in milliseconds before abandoning the attempt. If unspecified, defaults to platform-internal value.
Return value:
 None.
Exception:

Throws an exception when the call cannot be completed, including encountering busy signals or reaching a FAX or answering machine (Note: hardware may not support this feature).

5.2.6 Record

Record user audio to file.

Syntax:
 telephone.Record(url, endSilence, {maxTimeout}, {initialTimeout});
Parameters:
 o url: Required. The url of the recorded results.
 o endSilence: Required. Time in milliseconds to stop recording after silence is detected.
 o maxTimeout: Optional. The maximum time in seconds for the recording. Default is platform-specific.
 o initialTimeout: Optional. Maximum time (in milliseconds) of silence allowed at the beginning of a recording.
Return value:
 None.
Exception:
 . Throws an exception when the recording can not be written to the url.
5.3 Event Handlers
 App developers using telephone voice browser may implement the following event handlers.

5.3.1 onIncoming()

Called when the voice browser receives an incoming phone call. All developers can use this handler to read caller's address and invoke customized features before answering the phone call.

5.3.2 onAnswer()

Called when the voice browser answers an incoming phone call.

5.3.3 onHangup()

Called when user hangs up the phone. This event is NOT automatically fired when the program calls the Hangup or Transfer methods.

5.4 Example

This example shows scripting wired to the call control events to manipulate the telephony session.

```
<HTML>
<HEAD>
<TITLE>Logon Page</TITLE>
</HEAD>
<SCRIPT>
var focus;
function RunSpeech() {
  if (logon.user.value == "") {
    focus="user";
    p_u_d.start(); g_login.start(); dtmf.start();
  }
  return;
}
if (logon.pass.value == "") {
  focus="pin";
```

```
    p_pin.start(); g_login.start(); dtmf.start();
  }
  p_thank.start(); logon.submit();
}
function login_reco() {
  res = event.srcElement.recoResult;
  pNode = res.selectSingleNode("//uid");
  if (pNode != null)
    logon.user.value = pNode.xml;
  pNode = res.selectSingleNode("//password");
  if (pNode != null)
    logon.pass.value = pNode.xml;
}
function dtmf_reco() {
  res = event.srcElement.dtmfResult;
  if (focus == "user")
    logon.user.value = res;
  else
    logon.pin.value = res;
}
</SCRIPT>
<SCRIPT for="callControl" event="onIncoming">
<!-- // read address, prepare customized stuff if any
      callControl.answer();
//
</SCRIPT>
<SCRIPT for="callControl" event="onOffhook">
<!--
      p_main.start(); g_login.start(); dtmf.start();
      focus="user";
//
</SCRIPT>
<SCRIPT for="window" event="onload">
<!--
      if (logon.user.value != "") {
        p_retry.start();
        logon.user.value = "";
        logon.pass.value = "";
        checkFields();
      }
//
</SCRIPT>
<BODY>
  <reco id="g_login"
    onReco="login_reco(); runSpeech()"
```

```

timecut="5000"
onTimeout="p_miss.Start(); RunSpeech()" >
<grammar>
src=http://kokaneel/stradedemo/speechonly/login.xml/>
</reco>
<dtmf id="dtmf"
escape="#"
onKeyPress="g_login.Stop();"
onReco="dtmf_reco();RunSpeech()"
interdigitTimeout="5000"
onTimeout="dtmf.Flush(); p_miss.Start();RunSpeech()" />

<prompt id="p_main">Please say your user I D and pin
number</prompt>
<prompt id="p_uid">Please just say your user I D</prompt>
<prompt id="p_pin">Please just say your pin number</prompt>
<prompt id="p_miss">Sorry, I missed that</prompt>
<prompt id="p_thank">Thank you. Please wait while I verify
your identity</prompt>
<prompt id="p_retry">Sorry, your user I D and pin number do
not match</prompt>

<H2>Login</H2>
<form id="login">
  UID: <input type="text"
  onChange="runSpeech()" />
  PIN: <input type="password"
  onChange="RunSpeech()" />
</form>
</BODY>
</HTML>

```

6 Controlling dialog flow

6.1 Using HTML and script to implement dialog flow

This example shows how to implement a simple dialog flow which seeks values for input boxes and offers context-sensitive help for the input. It uses the title attribute on the HTML input mechanisms (used in a visual browser as 2

```

"tooltip" mechanism) to help form the content of the help
prompt.

<html>
<title>Context Sensitive Help</title>
<head>
<script>
var focus;
function RunSpeech() {
  if (trade.stock.value == "") {
    focus="trade.stock";
    p_stock.Start();
    return;
  }
  if (trade.op.value == "") {
    focus="trade.op";
    p_op.Start();
    return;
  }
  //... repeat above for all fields
  trade.submit();
}

function handle() {
  res = event.srcElement.recoResult;
  if (res.text == "help") {
    text = "Please just say";
    text += document.all[focus].title;
    p_help.Start(text);
  } else {
    // proceed with value assignments
  }
}

</script>
</head>
<body>
<prompt id="p_help" onComplete="checkFields()" />
<prompt id="p_stock" onComplete="g_stock.Start()">Please say
the stock name</prompt>
<prompt id="p_op" onComplete="g_op.Start()">Do you want to
buy or sell</prompt>
<prompt id="p_quantity" onComplete="g_quantity.Start()">How
many shares</prompt>
<prompt id="p_price" onComplete="g_price.Start()">What's the
price</prompt>

```

```

<reco id="g_stock" onReco="handle(); checkFields()" >
  <grammar src="/g_stock.xml" />
</ reco >

<reco id="g_op" onReco="handle(); checkFields()" />
  <grammar src="/g_op.xml" />
</ reco >

<reco id="g_quantity" onReco="handle(); checkFields()" />
  <grammar src="/g_quant.xml" />
</ reco >

<reco id="g_price" onReco="handle(); checkFields()" />
  <grammar src="/g_quant.xml" />
</ reco >

```

```

<form id="trade">
  <input name="stock" title="stock name" />
  <select name="op" title="buy or sell">
    <option value="buy" />
    <option value="sell" />
  </select>
  <input name="quantity" title="number of shares" />
  <input name="price" title="price" />
</form>
</body>
</html>

```

6.2 Using SMIL

The following example shows activation of prompt and reco elements using SMIL mechanisms.

```

<html xmlns:t="urn:schemas-microsoft-com:time"
      xmlns:sp="urn:schemas-microsoft-com:speech">
<head>
<style>
.time { behavior: url(#default#time2); }
</style>
</head>
<body>
  <input name="textBoxOrigin" type="text"/>
  <input name="textBoxDest" type="text" />

```

```

<sp:prompt class="time" t:begin="0">
  Please say the origin and destination cities
</sp:prompt>

<t:par t:begin="time.end" t:repeatCount="indefinitely">
  <sp:reco class="time" >
    <grammar src="/city.xml" />
    <bind targetElement="textBoxOrigin"
      value="//origin city" />
    <bind targetElement="textBoxDest"
      test="/smil/dest_city[@confidence >= 40]"
      value="//dest_city" />
  </sp:reco>
</t:par>
</body>
</html>

```

7 SMEX (Message) element/object

SMEX, short for Simple Messaging EXchange/EXtension, is an object that communicates with an external component or application on the platform of the client device. It can be embedded into an XML or similar markup based document as an element with the tag name <smex>. Exemplary usages of the messaging object may include logging and telephony controls. The object represents the extensibility of markup based recognition and prompting as it allows new functionality be added through messaging.

Upon instantiation, the object is directed to establish an asynchronous message exchange channel with a platform component or application through its configuration parameters or attribute specifications. It has a string property whose content is sent to the platform component or application whenever the property is the recipient of an

assignment operation (i.e., lvalue). Similarly, it also has a property of XML DOM Node type that holds the message received from the platform component or application. The message object sends an event whenever it receives a platform message. Since its basic operations are asynchronous, the object also has a built-in clock for the application developers to manipulate timeout settings.

The message or smex object is agnostic to the means of communications. However, in one embodiment, the smex object has the same life span as ordinary XML or markup elements, namely, the smex object will be destroyed when its hosting document is unloaded. While in many cases, the smex object can perform automatic clean-up and release communication resources when it is unloaded, there might be use cases (e.g., call controls) in which a persistent communication link is desirable across markup pages. For those cases, the architecture places the responsibility of relinquishing the allocated resources (e.g. close the socket) on the application developers.

The smex object is neutral on the format (schema) of messages. In some embodiments, it may be desirable to require implementers to support a few rudimentary schemes, with strong preferences to existing standard messages formats (e.g. as used in SIP or CCXML). In essence, the architecture allows both the platform and application developers to take the full advantage of the standardized extensibility of XML or similar markup to introduce other features without losing interoperability in the mean time.

Example 1: use of smex as a logging object

```
<smex id="logServer">
  <param name="d:server" xmlns:d="urn:Microsoft.com/COM">
    <d:protocol>COM</d:protocol>
    <d:clsid>20930930293202029320942098432098</d:clsid>
    <d:iid>0903859304903498530983309094803</d:iid>
  </param>
</smex>

<listen ...>
  ...// other directives binding reco results to input
  fields
    <bind targetElement="logServer" targetAttribute="sent"
      value="*[@log $ge$ 3]"/>
</listen>
```

This example demonstrates how a logging mechanism can be achieved using a COM object with its class id and interface id. The speech developers attach an attribute "log" indicating the level of interests for logging to the relevant SML nodes. In the example above, the app developer chooses to log all nodes with log value greater or equal to 3 by using a single bind directive. The example works in both downlevel and uplevel browsers.

The example also intends to demonstrate it is possible for a page to contain multiple smex objects communicated with the same platform component as long as there won't be confusion on which smex object is responsible for delivering the platform messages back to the recognition document. The above example implies a component can implement multiple interfaces, each of which has its own smex or message

conduit. The same argument applies to TCP servers listening to multiple ports.

Example 2: Reading the addresses for an incoming call:

```
<input type="text" id="remote"/>
<input type="text" id="transfer"/>
<input type="text" id="local"/>
<input type="hidden" id="session_id"/>
...
<smex id="telephone" sent="start_listening">
  <param name="server">http://tel-svr/whatever</param>
  <bind targetElement="session_id" value="//sid"/>
  <bind targetElement="remote" value="//remote_addr"/>
  <bind targetElement="transfer"
    value="//transfer_addr"/>
  <bind targetElement="local" value="//local_addr"/>
  ...
</smex>
```

This example shows how the bind directives can be used to process the received message. The example assumes a message for incoming call to have sub-elements remote_addr, transfer_addr, and local_addr whose contents represent the, remote, transfer, and local addresses of the incoming call, respectively.

In this example, a connectionless protocol based on HTTP is used to communicate with the telephony server. The telephony server here is designed to communicate with more than one browser instances, and hence each client must identify itself with a unique id assigned by the server when the app

starts. This is achieved in this example by sending the server a "start_listening" message. In this example, the session id is stored in a hidden field that can be sent back to the Web server and passed on to the next page of the application, although other techniques (e.g. Client side cookie) can also be used to manage session state. As in the case for reco, not every bind directives will be executed for every platform message. The above example does not imply the unique id will only be received when there is an incoming phone call.

7.1 Properties

The smex object can have the following properties, with only the Read/Write properties allowed to also serve as attributes for the initial value specification.

- sent: Read/Write, a string corresponding to the message to be sent to the platform component. Whenever sent is used as a lvalue, its contents are dispatched. There is no effect when the property is used as a rvalue or when a null object is assigned to this property.
- received: Read Only, an XML DOM Node data indicating the received message. The message will be available as a rvalue until the next onReceive event is ready to send.
- timer: Read/Write, a number in millisecond indicating the time span before a timeout event will be triggered. The clock starts ticking when the property is assigned a positive value. The value can be changed when a count down is in progress. A zero or negative value stops the clock without triggering the timeout event. The default is 0, meaning no timeout.

- **status:** Read Only, an integer indicating the recent status of the object. The possible values are 0, -1, and -2, which means normal, timeout expired, and communication with the platform cannot be established or has been interrupted, respectively. Platform specific error messages should be conveyed through the received property. For the cases that the error message is successfully delivered, the status code is 0.

7.2 Events

The object has the following events:

- **onReceive:** This event is sent when a platform message has arrived. If there are any directives declared by the bind elements, those directives will first be evaluated before the event is fired. Prior to the sending, the received property will be updated.
- **onError:** This event is sent when the timeout expires, or a communication link error has been encountered. When the event is sent, the **status** property will be updated with a corresponding error code as described above.

7.3 Child Elements

When assuming an element form, smex may have the following child elements:

- **bind:** same as in reco, except the directives are operated on the received message.
- **param:** same as in reco, provides platform specific parameters for the smex object. Each param element may be named using a "name" attribute, with the contents of the param element being the value of the parameter. In

one embodiment, the element should understand standard XML attributes for name space and XML data type declaration.

7.4 Other Comments

One elegant way to extend SMEX for the logging function would be

```
<smex id="logServer" ...> ... </smex>
<script> function logMessage(logClass, message) {
    logServer.sent = logClass + "|" + message;
} </script>
```

This in effect extends the object with a (global) function whose behavior can be customized. In the above example, the logging function is programmed to insert a field separator "|" between the id and the message.

For those who do not like global functions, they can use the "prototype" property of ECMAScript to attach the function as an object method. For example

```
<smex id="logServer" onload="addFunction()"> ... </smex>
<script>
    function my_logMessage(logClass, message) {
        logServer.sent = logClass + "|" + message;
    }
    function addFunction() {
```

```

logServer.prototype.logMessage =
my_logMessage;
}
</script>

```

One can refer to the function in a more objected-oriented manner:

```

logServer.logMessage(RECO_LOG_ERROR, "My
message");

```

It is noted that more work is required from the smex object implementers in order to make the extension work as in the above example, although all the necessary mechanisms are already well established standards.

4. Brief Description of the Drawings

FIG. 1 is a plan view of a first embodiment of a computing device operating environment.

FIG. 2 is a block diagram of the computing device of FIG. 1.

FIG. 3 is a plan view of a telephone.

FIG. 4 is a block diagram of a general purpose computer.

FIG. 5 is a block diagram of an architecture for a client/server system.

FIG. 6 is a display for obtaining credit card information.

FIG. 7 is a page of mark-up language executable on a client.

FIG. 8 is an exemplary page of mark-up language executable on a client having a display and voice recognition capabilities.

FIGS. 9A and 9B are an exemplary page of mark-up language executable on a client with audible rendering only and system initiative.

FIG. 10A and 10B are an exemplary page of mark-up language executable on a client with audible rendering only and mixed initiative.

FIG. 11 is an exemplary script executable by a server side plug-in module.

FIG. 12 is a pictorial illustration of a first operational mode of a recognition server.

FIG. 13 is a pictorial illustration of a second operational mode of the recognition server.

FIG. 14 is a pictorial illustration of a third operational mode of the recognition server.

FIGS. 15A and 15B are an exemplary page of declarative mark-up language executable on a client without scripting.

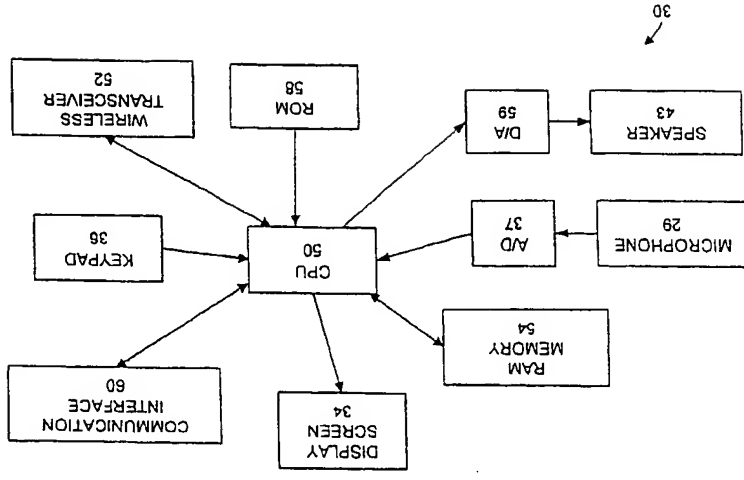


FIG. 2

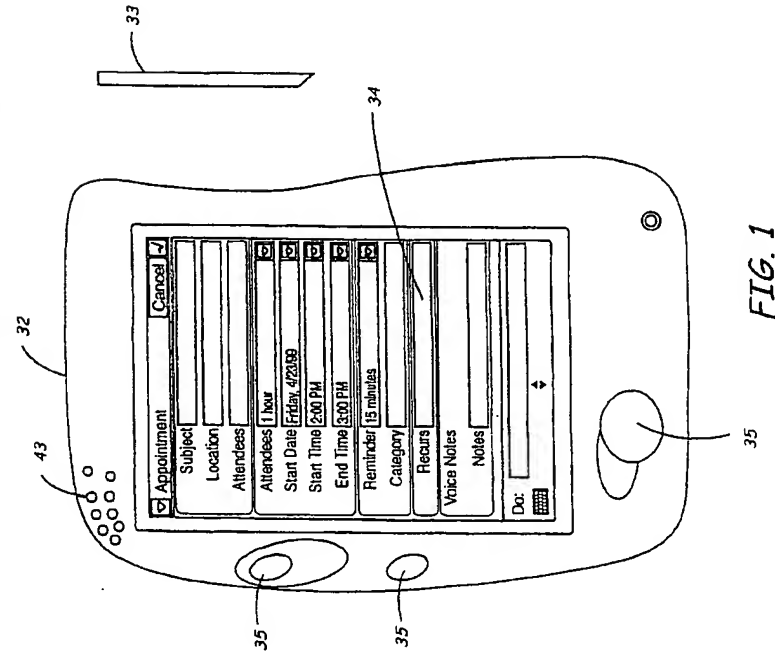
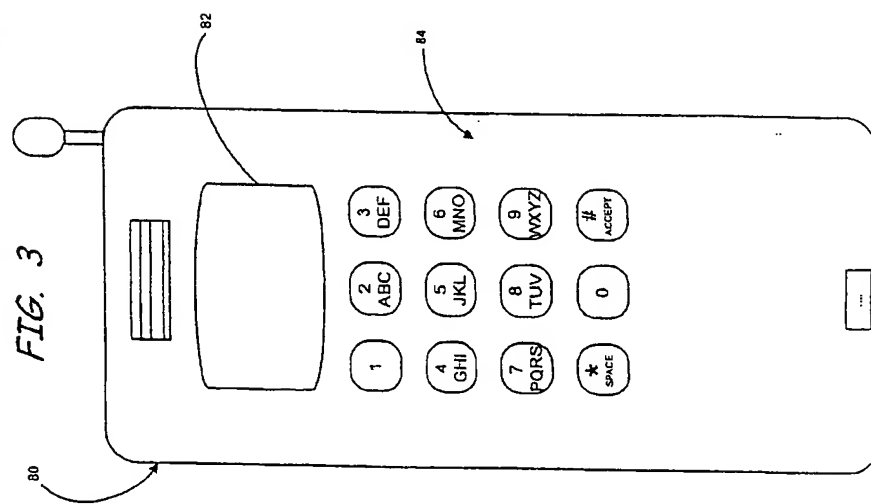
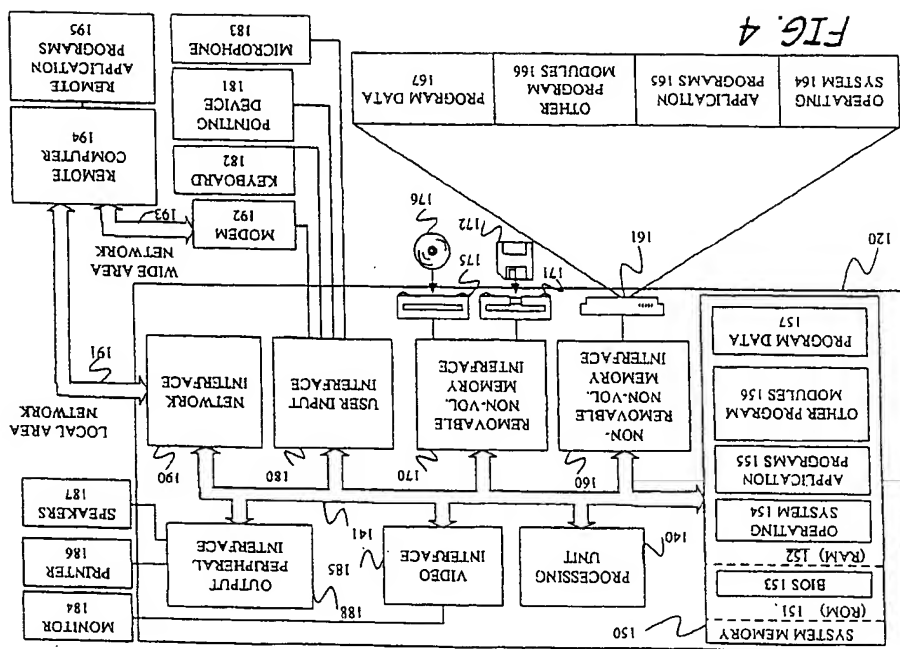


FIG. 1



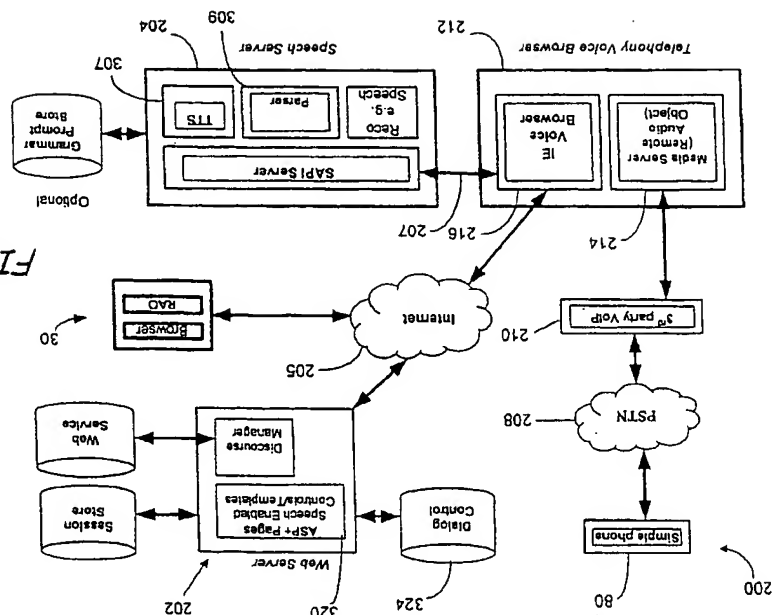
EXPIRATION DATE	254
CREDIT CARD NUMBER	252
CREDIT CARD TYPE	250
SUBMIT 264	

FIG. 6

```
<html>
<form id="get_card_info" method="post" action="http://payment.asp">
  <select name="card_type">
    <option value="amex">American Express</option>
    <option value="visa">Visa</option>
    <option value="mc">MasterCard</option>
  </select>
  <input type="text" name="card_num" width="30"
    onChange="handle()" />
  <input type="text" name="expiry_date" />
  <input type="submit" value="Submit" onClick="verify()" />
</form>
<script><![CDATA[
function handle() {
  if (get_card_info.card_type.value == "amex") {
    if (get_card_info.card_num.length != 15) {
      alert("amex should have 15 digits");
    }
  } else {
    if (get_card_info.card_num.length != 16) {
      alert("visa and master should have 16 digits");
    }
  }
  function verify() {
    var flag = window.confirm("submit the credit card info?");
    if (flag) {
      get_card_info.submit();
    }
  }
}]]>
</script>
</html>
```

FIG. 7

FIG. 5




```

function _handleO (
  handleO;
  checkFilled();
  if (card_type.value == "") {
    p_card_type.activate(); return;
  }
  if (card_num.value == "") {
    p_card_num.activate(); return;
  }
  if (expiry_date.value == "") {
    p_expiry_date.activate(); return;
  }
  p_content.activate();
  p_confirm.activate();
  confirmation.activate();
}

function confirmed(gob) {
  if (gob).recognizeText == "yes"
    getCardInfo.submit();
}

// user codes start here
function handleO {
  if (getCardInfo.card_type == "amex") {
    if (getCardInfo.card_num.length != 15) {
      prompt.speak ("amex should have 15 digits");
      getCardInfo.card_num = "";
    }
  }
  if (getCardInfo.card_num.length != 16) {
    prompt.speak ("credit card number should have 16 digits");
    getCardInfo.card_num = "";
  }
}
</script>
</body>

```

FIG. 9B

```

<body>
  <form id="get_card_info" method="post" action="http://paymentasp"
    onactivate="welcome();" >
    <prompt id="p_welcome">We now need your credit card</prompt>
    <prompt id="p_mumble">I didn't understand you</prompt>
    <prompt id="p_card_type" bargain="true">What credit card would you
      use?</prompt>
    <prompt id="p_card_num" bargain="true">Please say the number</prompt>
    <prompt id="p_expiry_date" bargain="true">What is the expiration
      date?</prompt>
    <prompt id="p_confirm">
      I have your <value select="card_type" /> <value select="card_num" />
      with expiration date <value select="expiry_date" />
    </prompt>
    <prompt id="p_confirm">Is this correct?</prompt>
    <reco id="g_get_card_info" onReco="_handle()" onNoReco ="mumble(this)">
    <grammar src ="/gram/getPayment" />
    <bind target="card_type" value="/card/type" />
    <bind target="card_num" value="/card/number" />
    <bind target="expiry_date" value="/card/exp_date" />
405 </reco>
    <reco id="confirmation" onReco ="confirmed(this) onNoReco ="mumble(this)" />
    <grammar src ="/gram/yesno" />
    <select name="card_type">
      <option value="amex">American Express</option>
      <option value="visa">Visa</option>
      <option value="ms">MasterCard</option>
    </select>
    <input type="text" name="card_num" width="30" />
    <input type="text" name="expiry_date" />
    <input type="submit" value="Submit" />
    </form>
    <script><![CDATA[
      function welcomeO {
        p_welcome.activate();
        repeat = 0;
        checkFilled();
      }
      function mumble(gob) {
        p_welcome.deactivate();
        p_mumble.activate();
        checkFilled();
      }
      function _handleO {
        handleO;
        checkFilled();
      }
    ]]>

```

FIG. 10A

TO FIG. 10B

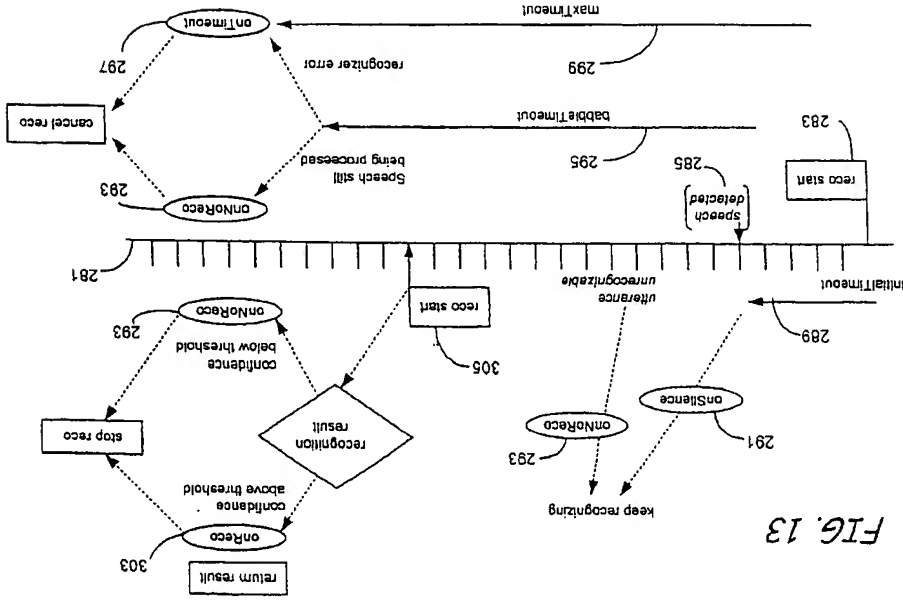


FIG. 13

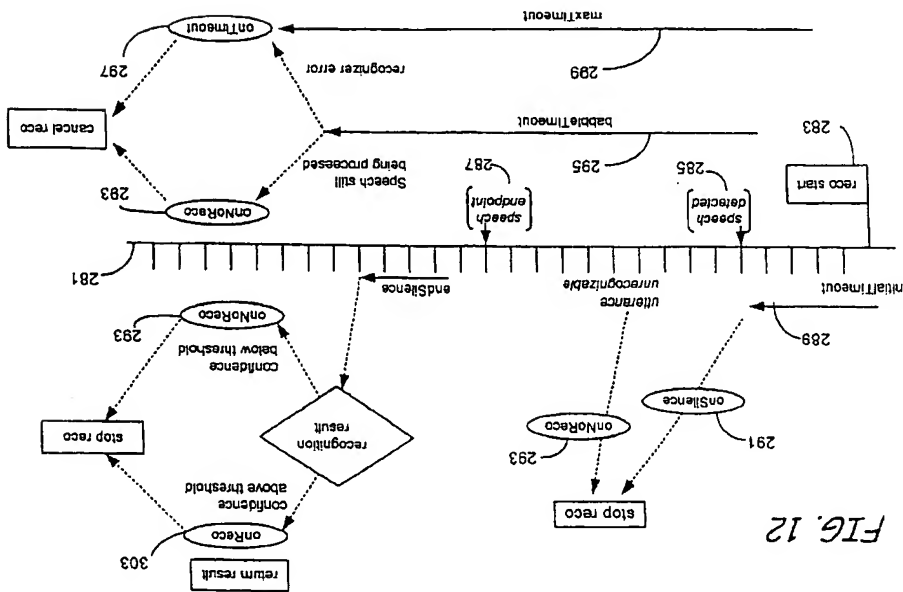


FIG. 12

```

350 <html>
351 <body>
352 <!-- the data section -->
353 <form id="get drink">
354 <input name="drink" />
355 <input type="radio" name="cream" />
356 <input type="radio" name="sugar" />
357 <input name="uid" type="hidden" />
358 </form>
359 <!-- The speech section -->
360 <prompt id="welcome">Welcome, caller! </prompt>
361 <prompt id="ask">Do you want coke, coffee, or orange juice?
362 </prompt>
363 <prompt id="confirm">I heard <value href="drink"/>. Is this
364 correct? </prompt>
365 <prompt id="thanks">Thank you. Please wait when I get it for
366 you. </prompt>
367 <prompt id="retry">Okay, let's do this again </prompt>
368 <prompt id="reprompt">Sorry, I missed that. </prompt>
369 <prompt id="cream_sugar">No you want cream or sugar with your
370 coffee? </prompt>
371 <reco id="reco_drink"><grammar src="/drink.xml"/>
372 <bind test="/@confidence >$ 10"
373 <targetElement="reco_confirm" targetMethod="start"
374 <targetElement="reco_drink" targetMethod="start"/>
375 <targetElement="reco_confirm" targetMethod="start"/>
376 <bind test="/@confidence >$ 10"
377 <targetElement="reco_drink" values="drink"
378 <targetElement="reco_confirm" targetMethod="start"
379 <targetElement="reco_confirm" targetMethod="start"
380 </reco>

```

FIG. 15A

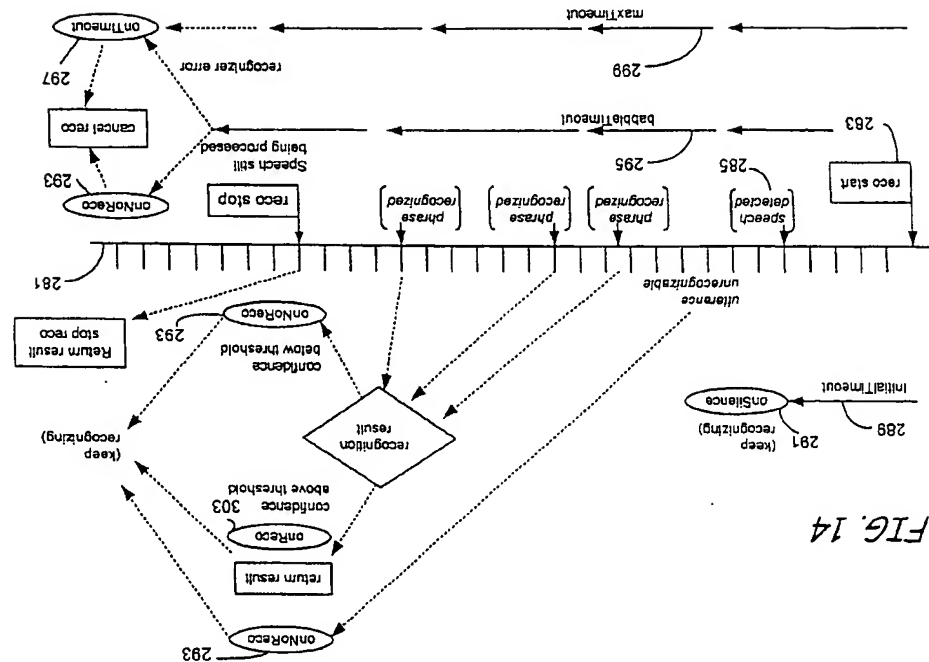


FIG. 14

```

350 <reco id="reco_cream_sugar"><grammar src="/cream+sugar"/>
    <bird test="/@confidence $g1 $10 and
    host()/get_drink="cream" targetAttribute="checked"
    targetElement="cream" targetAttribute="checked"
    value="/cream/@value"
    targetElement="sugar" targetAttribute="checked"
    value="/sugar/@value"
    targetElement="confirm" targetMethod="start"
    targetElement="reco_yesno" targetMethod="start"/>
355 </reco>

360 <reco id="reco_yesno"><grammar src="/yesno"/>
    <bind test="/yes/@confidence $g1 $10"
    361 targetElement="thanks" targetMethod="start"
    362 targetElement="get_drink"
    363 targetMethod="submit" />
    <bind test="/no or /@confidence $les 10"
    364 targetElement="raty" targetMethod="start"
    365 targetElement="ask" targetMethod="start"
    366 targetElement="reco_drink"
    367 targetMethod="start"/>
368 </reco>

369 <!--call control section -->
    <smex id="telephone" sent="start_listening"><param
    server="cxmproc"> </param>
    <bind targetElement="uid" value="/@uid"/>
    <bind test="/call_connected"
    370 targetElement="welcome" targetMethod="start"
    371 targetElement="ask" targetMethod="start"
    372 targetElement="reco_drink"
    373 targetMethod="start"/>
    </smex>
    </body>
    </html>

```

FIG. 15B

1. Abstract

A markup language for execution on a client device in a client/server system includes extensions for recognition.

2. Representative Drawing

FIG. 1